

ON THE APPLICATION OF CONCEPTUAL CLUSTERING FOR
KNOWLEDGE DISCOVERY IN DATABASE SYSTEMS

By

TAREK M. ANWAR

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF
PHILOSOPHY

UNIVERSITY OF FLORIDA

1992

In the name of Allah, Most Gracious, Most Merciful
High above all is Allah, the King, the Truth! Be not in haste with the Quran before
its revelation to thee is completed, but say, 'O my Lord! increase me in knowledge'

(20, 114)

I dedicate this thesis to my father, Dr. Mohammed Sami Anwar, and my family
for their continuous support throughout my graduate studies.

ACKNOWLEDGMENTS

The author wishes to extend his most sincere gratitude to all who assisted him during his study. My sincere appreciation goes out to Dr. Shamkant Navathe for being the chairman of my supervisory committee, and for his guidance, assistance and patience throughout my graduate studies. Sincere thanks are also extended to Dr. Jose Principe for his valuable and insightful discussions on various technical and philosophical issues. The author would especially like to express his utmost gratitude to Dr. Howard Beck for his invaluable kindness, guidance, assistance, patience and direction throughout the program. I am deeply grateful to Dr. Herman Lam for valuable technical discussions throughout my graduate studies. I thank Dr. Sharma Chakravarthy for fruitful technical discussions and for access to his personal library. I would like to thank Dr. Keith Doty for his time and for agreeing to serve on my supervisory committee. Finally, my special thanks go to Sharon Grant whose help has been simply immense.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	ii
LIST OF FIGURES	vi
ABSTRACT	vii
CHAPTERS	
1 INTRODUCTION	1
1.1 Schema Integration	3
1.2 Schema Evolution and Exception Handling	4
1.3 Inexact Querying	5
2 APPROACHES TO CLUSTERING	7
2.1 Applications of Cluster Analysis	10
2.2 Components of Cluster Analysis	11
2.2.1 Choice of Data Objects	12
2.2.2 Choice of Attributes	12
2.2.3 Homogenization of Attributes	14
2.2.4 Similarity Measures	16
2.2.5 Clustering Criteria	17
2.3 Quantitative Similarity Measures	20
2.3.1 Distances	20
2.3.2 Probabilistic Approaches	22
2.3.3 Fuzzy Sets	26
2.4 Qualitative Similarity Measures	29
2.4.1 A Theory Of Categorization	30
2.4.2 Conceptual Clustering Techniques	33
3 SCHEMA INTEGRATION THROUGH CONCEPTUAL CLUSTERING	38
3.1 Conventional Schema Integration Methodologies	38
3.2 Architecture of a Conceptual Clustering System	43
3.2.1 System Architecture	43
3.2.2 The CANDIDE Data Model	52
3.3 Application of the Conceptual Clustering System to Databases	55
3.3.1 Determination of the Relevant Instance Set and Relevant Attribute Set	57
3.3.2 Generation of the Initial Class Taxonomy and Derived Attribute Set	61

3.3.3	Determination of Attribute Restrictions and Final Class Taxonomy	64
3.4	Implementation Issues	67
4	SCHEMA EVOLUTION AND EXCEPTION HANDLING TECHNIQUES	72
4.1	Comparison to Related Work	73
4.2	Exception Handling	77
4.3	Incremental Clustering Algorithm	80
4.3.1	Generation of the Immediate Description Graph (IDG)	82
4.3.2	Generation of the Extended Description Graph (EDG)	83
4.3.3	The Graph-Matching Technique	85
4.3.4	Class Description Generation From EDG	89
4.3.5	Schema Evolution	91
4.4	Performance	91
5	IMPRECISE QUERYING	93
5.1	Comparison to Related Work	96
5.2	Imprecise Querying Algorithm	99
5.2.1	Generalization of Cardinality Constraint	101
5.2.2	Generalization of Value Set Constraints	104
5.2.3	Example	106
5.3	Using the Database Taxonomy to Determine Relative Similarity	108
5.4	Performance Issues	110
6	SUMMARY AND CONCLUSIONS	111
	REFERENCES	115
	BIOGRAPHICAL SKETCH	120

LIST OF FIGURES

2.1	The set of data objects.	25
2.2	Binary attribute discrimination based on 'height'.	25
2.3	Final decision tree.	27
3.1	Conceptual Clustering System Architecture.	44
3.2	The background knowledge base. a) A semantic network representing interrelationships between attributes; b) Dataflow graph of <i>salary = hrs-worked × wages-per-hr.</i>	49
3.3	A CANDIDE class taxonomy and attribute hierarchy. a) Class taxonomy; b) Attribute hierarchy.	56
3.4	Instances of Database-1	58
3.5	Instances of Database-2	59
3.6	Relevant instances	62
3.7	The initial taxonomy	64
3.8	Instances of COURSE referenced by 'teaches'	67
3.9	The final taxonomy	68
3.10	The generated class descriptions	69
3.11	The compiled relevancy network for the previous example	70
3.12	The organization of the database	70
4.1	Main components of the conceptual clustering algorithm	81
4.2	Example database instances.	82
4.3	A graphical representation of the instance Jim	84
5.1	A schematic illustration of the imprecise querying algorithm	100

5.2	A CANDIDE class taxonomy and attribute hierarchy. a) Class taxonomy; b) Attribute hierarchy	102
5.3	A lattice of generalized queries	107

Abstract of a Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

ON THE APPLICATION OF CONCEPTUAL CLUSTERING FOR
KNOWLEDGE DISCOVERY IN DATABASE SYSTEMS

By

Tarek M. Anwar

August 1992

Chairman: Dr. Shamkant B. Navathe
Major Department: Electrical Engineering

Knowledge discovery is the nontrivial and efficient extraction of high-level patterns from databases. In this thesis we propose an approach to knowledge discovery in terms of discovery of class formations from a set of existing instances of data. The emphasis in this approach is on reasoning at the instance level with which we are able to generate classes, and a schema that more accurately and precisely reflects the actual data stored rather than ad hoc class formations. Specifically, three *attribute-based purpose-directed conceptual clustering* techniques are presented. These techniques are not confined to any particular semantic data model but can easily be adapted to any data model.

A conceptual clustering technique for database integration is introduced by which schema generation occurs by conceptually clustering the underlying data instances of several (possibly heterogeneous) databases. This process is guided by specifying a context in the form of a *clustering seed*. By modifying the clustering seed we are able to vary the schema generated to accommodate different user groups needs.

A second conceptual clustering technique is utilized for schema evolution and exception handling. This technique determines instance similarity through a graph-matching procedure. A class description is subsequently generated and is incorporated into the schema. Moreover, this technique provides a class taxonomy and thereby an intensional answer to a query that is conceptually more informative than a set of instances. A key issue in schema evolution is that of exception detection in which this algorithm aids.

Finally, a conceptual clustering technique for imprecise querying is detailed. In contrast to numeric or fuzzy sets approaches which ultimately rely on some distance metric and threshold to processing such queries, conceptual clustering retrieves instances which are *structurally, semantically, and pragmatically* similar to the query even though they may not match the requirements exactly. The query processor has both a deductive and inductive component. The deductive component finds precise matches in the traditional sense, and the inductive component identifies ways in which imprecise matches may be considered similar. Ranking on similarity is done using the database taxonomy, by which similar instances become members of the same class. Relative similarity is determined by depth in the taxonomy.

Overall, this thesis has applied machine learning techniques (learning from observation) that are based on psychological principles of category formation to the difficult problems of schema design, evolution and integration, and imprecise querying in database systems.

CHAPTER 1 INTRODUCTION

One of the basic characteristics of a database management system is that it utilizes a nonredundant, unified representation of all the data it manages, called the schema [6, 18]. The desire to elaborate and enrich the expression of this type of representation led to the introduction of semantic data models [29]. Semantic data models provide richer modeling capabilities than traditional record-oriented data models of the network, hierarchical or relational type. In particular, research in this area has articulated a number of constructs for representing complex interrelationships among data [29]. A construct is commonly used in all these semantic data models to represent a generic concept. This construct is referred to by many names such as a class [29], an entity type [18], a category [19], or a frame [12]. In this thesis we will refer to this construct as a class, unless otherwise specified.

A class models a generic concept as an abstraction of a set of instances which manifest that generic concept. These instances are members, the *extension*, of the class. A class may also contain an internal description, the *intension*. This description identifies other concepts associated with the class through different types of relationships (such as generalization, specialization, aggregation, association) and conditions (semantic integrity constraints) that dictate this type of association. Through certain types of relationships (generalization and specialization) some data models allow classes to inherit properties from one or more other classes. In some data models class membership is determined by *necessary and sufficient conditions* as exemplified by the Entity-Category-Relationship data model [18] where predicate-defined categories can be specified and in term-subsumption languages such as KL-ONE [12] or CANDIDE [8] where attribute restrictions may be used. If an instance satisfies these

conditions, it is automatically added to the class. In other words, the generic concept modeled by that class is represented as a *predicate* defined over instances. However, in most data models, class membership is determined manually without the use of necessary and sufficient conditions. Essentially, a schema of a semantic data model consists of a set of data model constructs assembled together using some modified version of a semantic network [18, 29].

It is the view of this thesis that class formation in semantic data models is ad hoc due to the varied treatment of classes and because the issue of grouping instances into classes is considered an art rather than a science. Currently, the process of generating class descriptions is very difficult due to the unpredictability and evolution of the real world and the error-prone process of interviewing end-users to determine relevant classes, attributes and relationships [10]. Moreover, the expressive power of semantic data models gives designers greater leeway in modeling an application domain without any guidance as to the appropriateness of the constructs used. That is, an application domain can be modeled in several different ways with different combinations of constructs without any guidelines as to the quality of the schema generated. This is termed *relativism* [26]. Therefore, the design process can be characterized by a certain amount of indeterminism in choosing the constructs [11]. Consequently, the quality of the classes generated, and hence the schema are dependent on the designer's intuition and experience.

In addition to this, it is extremely difficult, if not impossible, to define 'natural kinds' using necessary and sufficient conditions as utilized in some data models. How can you define a 'university student' using necessary and sufficient conditions? Is a 'university employee' enrolled in one course also a 'university student'? Because of these reasons a database class is usually too general and does not specify all the pertinent information of the corresponding generic concept. Class definitions that do

not accurately specify the generic concept modeled by that class lead to exceptions. By an exception we mean an instance that belongs in a certain class but does not meet or satisfy that class description. The converse may also be true in that an instance that does not belong to a class may satisfy that class description [7, 10, 34]. The overwhelming majority of database systems are rigid in the sense that they do not tolerate exceptions. Exception handling duties are delegated to the user who is forced to *tailor the data to the schema* resulting in a possible loss of information.

In this thesis we view schema design as a knowledge discovery process by which class formations and schema are generated through the use of machine learning techniques. Specifically, these techniques create *conceptually cohesive* [36] classes that are based on psychological principles of human category theory rather than ad hoc classes. Schema generation occurs as a result of conceptually clustering the underlying data instances. Three conceptual clustering techniques will be introduced for schema integration, schema evolution and exception handling, and query processing. That is, all these three database functions will be considered as a form of schema design. For the generic use of this methodology in different contexts we shall use the term **view generation** to refer to the activity of defining a schema over single or multiple databases.

1.1 Schema Integration

A conceptual clustering system (CCS) for schema integration is introduced. The CCS consists of four main modules. These modules are the databases (to which it is connected), the background knowledge base, the clustering seed, and the view generator. The background knowledge base of the system consists of a set of facts that model several types of interrelationships between instance attributes. The clustering seed guides the conceptual clustering process by specifying the relevant attributes of

the view desired and the inductive preference criteria. The view generator will then generate views according to the clustering seed, background knowledge and, database instances. Hence, schema integration occurs as a result of conceptually clustering the underlying data instances and guiding this process by specifying a clustering seed. By reasoning at the instance level we are able to generate classes and schema that more accurately reflect the actual data stored. By contrast, conventional integration methodologies are data model dependent and integrate at the construct level. These methodologies reason at the class level to the exclusion of instances. Moreover, these methodologies rely on simple measures of similarity such as string and domain comparisons which are incapable of capturing essential interrelationships between database objects vital to the integration process.

1.2 Schema Evolution and Exception Handling

An important service to be provided by a database system would be the ability to refine its schema based on the data stored so far. Such a service is termed schema evolution. Today's commercial systems totally lack such a facility. An incremental conceptual clustering algorithm for schema evolution is presented. This algorithm aids in building classes by grouping related instances and developing class descriptions. An instance may be assigned to a class either because it meets a class description or because it is similar to other instances in that class. Class cohesion results from a trade-off between class descriptions and instance similarity. Instance similarity is determined by a function, INTERSECT, which utilizes a graph-matching technique. Moreover, INTERSECT is used in defining an exception condition. Exception handling results in schema modification. This is in contrast to other techniques that simply add the description of an exception to the class description. Such systems are

not dynamic and are still unable to handle exception which fail to satisfy either the typical description or the extended description containing 'known exceptions'.

1.3 Inexact Querying

A facility for inexact querying has been implemented which uses conceptual clustering techniques. In contrast to numeric [30, 40] or fuzzy set [52] approaches which ultimately rely on distance metrics and thresholds to process such queries, conceptual clustering retrieves instances which are structurally (based on correspondence between object components), semantically (due to correspondences in meaning), and pragmatically (in terms of relevancy to the goal, query object) similar to the query even though they may not match the query exactly [28]. The query processor has a deductive and inductive component. The deductive component finds exact matches in the traditional sense by retrieving sets of instances that completely satisfy the conditions specified in a query. Consequently, query processing is based on deductive inferencing about structured objects rather than procedural specification of operations. Inferencing techniques are defined formally by the subsumption function. Query specification is entirely declarative; the user need not specify *how* (in terms of algebraic operations) or from *where* (logical access paths and exact attribute names) the query is to be executed. The inductive component identifies ways in which inexact matches may be considered similar. The inductive component generates new class descriptions from the original query. This is done by generalizing the constraints specified in the query in different *directions* and to different *levels*. The clustering algorithm thus automatically generates a class taxonomy and groups instances into classes. The imprecise query algorithm is triggered by a null answer in response to a query. That is, if no instances in the database satisfy the conditions stated in the query, the system will try to find instances that partially satisfy the query.

In a nutshell, the goal of the present thesis is to correct the shortcomings and limitations of conventional schema design methodologies mentioned earlier as follows:

- Generate conceptually cohesive classes based on category theory rather than do ad hoc class formations.
- Recognize data instances in class formation and incorporate them in the process of generating more accurate class descriptions.
- Capture essential semantics in defining schemas so as to avoid ad hoc class formations due to simple and inadequate measures of similarity.
- Generate the primary constructs of semantic data models. Thus, our methodology is not geared toward one particular semantic data model but can be easily tailored to any data model.
- Reduce user interaction in the process of schema design because of the formal specification of classes.

The remainder of the thesis is organized as follows. In the next chapter the components of clustering algorithms are discussed and specific clustering approaches are reviewed. Specifically, the types of similarity measures used by these different approaches are compared and contrasted. After defining class formation by conceptual clustering in chapter 2, chapter 3 details a conceptual clustering algorithm utilized for schema integration. The schema evolution and exception handling techniques are presented in chapter 4. Chapter 5 illustrates the imprecise querying algorithm. Concluding remarks are given in chapter 6.

CHAPTER 2 APPROACHES TO CLUSTERING

One of the basic and most frequent activities of humans is that of sorting similar stimuli or objects into categories. The various stimuli encountered by a human during a single day are simply too numerous for mental processing as unique objects. Instead, each stimuli is described primarily in terms of category membership. This ability of humans to make accurate generalizations from a seemingly random set of stimuli or to discover patterns in them is a fascinating research topic of long standing interest. It is this ability that holds the key to an improvement of the methods by which computers can acquire knowledge. A need for such an improvement is evidenced by the fact that knowledge acquisition is presently the most limiting 'bottleneck' in the development of modern knowledge-intensive artificial intelligence systems. As mentioned by Lazarsfeld and Reitz,

The use of any knowledge reaches into three areas of the mind; the search for truth, the skill of forecasting, and the gift to imagine a future different from the present. There will never be clear-cut rules of procedure. The best we can hope for is to *sort out the different strands and look for regularities as they are combined into cords*. The more we succeed the more energy and freedom will be left for the creative and innovating element indispensable in all utilization. ([33], p. 14.)

Inductive learning is such a technique that can help one 'look for regularities.' The above ability is achieved by a process called *induction*, that is inductive inference from facts provided by a teacher or the environment. The studies and modeling of this form of learning is one of the central topics of machine learning. The goal of induction is to formulate a general assertion that implies the given object descriptions. In contrast to deduction, the starting premise of induction are specific facts rather than axioms. For any given set of facts, there will be a potentially large number of hypotheses that could be generated that imply these facts.

Philosophers have devoted a lot of attention to the validity of inductive inferencing but have been only partially successful in this task [22]. The philosopher Francis Bacon pointed out that an inductive leap, to be of value, must go further than the facts warrant. When it does, and is subsequently confirmed by observation, our confidence is strengthened.

But in establishing axioms by this kind of induction, we must also examine and try whether the axioms so established be framed to the measure of those particulars only from which it is derived, or whether it be larger and wider. And if it be larger and wider, we must observe whether by indicating to us new particulars it conforms that wideness and largeness as by a collateral security; that we may not either stick fast in things already known, or loosely grasp at shadows and abstract forms; not at things solid and realised in matter. ([22], p. 16.)

Bertrand Russell also wrestled with the problem of inductive reasoning and eventually admitted defeat. The inductive principle is 'incapable of being proven by an appeal to experience,' ([22], p. 17.) claimed Russell. But, he also adds that 'we just either accept the inductive principle on the grounds of its intrinsic evidence, or forgo all justification of our expectations about the future' ([22], p. 17.). In other words, if you don't believe in induction you do not believe in anything.

Despite these philosophical problems, there has been extensive research in the area of inductive learning and cluster analysis has evolved as one of the few proven systematic techniques of inductive learning. In cluster analysis, little or nothing is known about the category structure. All that is available is a collection of observations whose category memberships are unknown. Classification, or identification, is the process or act of assigning a new item or observation to its proper place in an established set of categories. The classification problem may be complicated by imperfect class definitions, overlapping categories and random variations in the observations.

The operational objective of cluster analysis is to discover the category structure which fits the observations. The problem is frequently stated as one of finding the 'natural groups.' In a more concrete sense, the objective is to sort the observations in groups such that the degree of 'natural association' is high among the members of the same category and low between members of different groups. In essence, the question is to determine what are these 'natural groups' and 'natural associations' [1, 46]. Even though little or nothing is known about the category structure beforehand, one frequently has some latent notions of desirable and unacceptable features of a classification scheme. So, why not enumerate all the possibilities and simply choose the most appealing? The number of ways to sort n observations into m groups ($m \leq n$) is a Stirling number of the second kind [1]. That is,

$$S_n^{(m)} = \frac{1}{m!} \sum_{k=0}^{k=m} (-1)^{m-k} \binom{m}{k} k^n$$

For even a relatively simple problem of sorting just 10 objects into 5 categories, the number of possibilities is

$$S_{10}^{(5)} = 42,525.$$

The problem is usually compounded by the fact that the number of groups is usually unknown, so that the number of possibilities is the sum of Stirling numbers. In the case of 10 observations

$$\sum_{j=1}^{j=10} S_{10}^{(j)} = 115,975.$$

Which is a large number indeed for such a simple problem. Certainly, most of these observations are just minor variations of some more appealing category structure. In any case, the human mind generates a series of shortcuts that permits a reasonable arrangement in little or more time than it would take to put everything in one or more category, without regard to similarity or compatibility among items. It is generally

the intent of cluster analysis algorithms to emulate such human efficiency and find an acceptable solution while considering only a small number of alternatives [1, 46].

2.1 Applications of Cluster Analysis

Cluster analysis is a tool for suggestion and discovery. It is not itself a wellspring of either truth or falsehood. Cluster analysis has been employed as an effective tool in scientific inquiry. One of its most valuable roles is to generate hypotheses about category structure. Put another way, cluster analysis may be used to reveal structure and relations in data. It is a tool of discovery. Obviously, such methods are more suitable and efficient in large problems than humans are. The result of cluster analysis can contribute directly to development of inductive generalizations. More specifically, the application of cluster analysis has been evident in several major fields.

1. *Life sciences.* The operational purpose of cluster analysis in these fields (such as biology, zoology, etc) may range from developing complete taxonomies to delimiting the subspecies of a distinct but varied species. Typically, data objects represent several different life forms such as plants, animals, insects and microorganisms.
2. *Medical sciences.* Cluster analysis is performed on data objects representing diseases, patients and symptoms. The operational emphasis here is on discovering more effective and economical means for making positive diagnoses in the treatment of patients.
3. *Engineering sciences.* Typically, clustering is used to help perform pattern recognition. Typical examples of the entities to which clustering has been applied to include hand written characters, samples of speech, pictures, random signals, etc.

4. *Artificial Intelligence.* The present approach to constructing knowledge bases involves the tedious process of formalizing the experts' knowledge and encoding it in some knowledge representation system, such as production rules or semantic networks. Clustering techniques could provide both an improvement of the current techniques and a basis for developing alternative knowledge acquisition techniques. In a less direct, but potentially promising, use of cluster analysis is for the refinement of knowledge bases initially developed by human experts. Here, clustering techniques could be used to rectify inconsistencies, to remove redundancies and to simplify expert-driven decision rules.

The rest of the chapter is organized as follows. In the next section we present the basic components of clustering techniques. Section 2.3 discusses several of the conventional quantitative similarity techniques. Qualitative similarity measures are presented in Section 2.4 along with a category theory on which they are based.

2.2 Components of Cluster Analysis

The notion of clustering is very intuitive, yet the actual considerations required to perform cluster analysis entail a host of problems. To begin with, the term cluster analysis does not refer to a single systematic technique, but rather a plethora of heuristic procedures. The actual search for clusters in real data involves a series of intuitive decisions as to which elements of known clustering techniques should be utilized [1, 46]. There appear to be five major components to clustering a set of data objects, all of which affect the outcome considerably. These are the choice of data objects to cluster, the choice of object attributes, homogenization of the attributes of the data objects, similarity measures between data objects and clustering criteria. These components are discussed in detail in the next five subsections.

2.2.1 Choice of Data Objects

There are two different situations of interest in the choice of data objects. In the first case, the set of data objects is complete. That is, only those data objects will be considered and the results of clustering are intended to be applied to those objects only. The purpose here is to discover a classification scheme for that given set of objects. Such is the case in the pattern recognition of written alphabetical characters, for example. The principal consideration is to make sure that no important data objects are missing.

In the second case the set of data objects is a sample of a larger population which is the true object of interest. This situation occurs more frequently and greater care must be taken. Obviously, a random sample of objects from the population will help in establishing an unbiased representation of the facts. Such is the case in knowledge acquisition techniques in which the domain of discourse is not limited and that certain refinements must be considered. Typically, such clustering techniques tend to be incremental, that is, are capable of incrementally modifying the clusters generated by the algorithm to accommodate new data objects.

2.2.2 Choice of Attributes

Data objects are described in terms of their properties and other attributes. It is probably the choice of attributes that has the greatest influence on the ultimate results of a cluster analysis. Attributes which are largely the same for all data units have little discriminating power whereas those manifesting consistent differences from one subgroup to the other can induce strong distinctions. When a relevant discriminating attribute is left out of the analysis, some clusters may emerge into a confusing mass with no distinctive boundary. On the other hand, inclusion of strong discriminators not particularly relevant to the purpose at hand can mask the sought for

clusters and give misleading results. In the quest for clusters, two possibilities should not be overlooked:

1. *The data may contain no clusters.* When data objects have an absence of discriminating attributes and a more or less uniform distribution of points in the measurement space would lead to a distinct lack of cohesion.
2. *The data may contain only one cluster.* In the clustering of data objects, the absence of discriminating attributes combined with meaningful mutual association among all data units would give rise to only one cluster.

Clearly these two possibilities are the extremes with all other possibilities falling in between.

In addition to the ability to evaluate observations, the knowledge of the context is fundamental to the conceptual power to explain causes and findings. *Context* is a collective term for all those elements of knowledge that pertain to our domain of discourse. As Hanson stated,

We have had an explanation of x only when we can set it into an interlocking pattern of concepts about other things, y and z. A completely novel explanation is a logical impossibility. It would be incomprehensible . . . ; it would be imponderable, like an inexpressible or unknowable fact. ([1], p. 20.)

Moreover, any given set of data may admit of many different but meaningful classifications. Each classification may pertain to a different aspect of the data. It is unnecessarily narrowing to seek a single 'right' classification. Several different clustering procedures will be needed to discover multiple classifications. Cluster analysis methods involve a mixture of imposing a structure on the data and revealing that structure which actually exists in the data. To a considerable extent the set of clusters reflects the degree to which the data set conforms to the structural forms embedded in the clustering algorithm.

2.2.3 Homogenization of Attributes

A prevalent obstacle in the clustering of data objects is the lack of homogeneity among their attributes. In measuring similarity (or association) between attributes it is necessary to establish a single common index of similarity. The contribution of each attribute to this index depends on its scale of measurement and that of other attributes. Some investigators recommend reducing all variables to a standard form at the outset. This is called *normalization*. Such suggestions simplify the mechanics of the analysis but constitute a complete abdication of the analyst's responsibilities and the possible loss of information [1, 46].

In the real world natural formulations frequently entail a mixture of attribute types. A systematic and comprehensive classification of attributes provides a convenient structure for identifying essential differences among data objects. Mathematicians tend to distinguish based upon the number of values in the range of the attributes (i.e. the value set), that is, the number of distinct values the attribute may assume. Thus, attributes are classified as follows.

1. *Continuous*. Attributes with a continuous scale have an uncountably infinite range. That is, the range of the attribute cannot be put into one-to-one correspondence with the set of positive integers. Practically speaking, this type of classification is only of theoretical importance to our considerations. Due to the limited resources of a computer, all the values in the range of an attribute can be enumerated and be put into a one-to-one correspondence with the set of positive integers.
2. *Discrete*. Attributes with this type of range have a finite, or at most a countably infinite, range. Hence, the range may be put into one-to-one correspondence with a subset, or the complete set, of positive integers.

In other scientific fields attributes are usually categorized based on the following scale of measurement.

1. *Nominal scale.* Values of a nominal scale can only be distinguished from each other. Consequently, one can only determine if two attribute values are equal or not. As an example, consider the attribute 'marital-status' with the value set $\{\text{married}, \text{single}\}$.
2. *Ordinal scale.* This type of scale not only distinguishes between its values but also induces *ordering*. Thus, one is capable of determining whether one value is greater or less than another. Consider the attribute 'age' with the associated value set $\{\text{child}, \text{teenager}, \text{adult}\}$. In this case one can determine that if a person is an *adult* he is older than one who is a *child*.
3. *Interval Scale.* This type of scale assigns a meaningful measure of difference between its values. Hence, we are not only capable of determining if one value is greater than another, but by *how much*. Consider the attribute 'tv-channel' with the associated value set $\{1, 2, 3, \dots, 32\}$.
4. *Ratio Scale.* This type of scale not only assigns a meaningful measure of difference between its values but also has a meaningful *reference point* or *origin*. Distances are classical examples of this type of scale. Consider the attribute 'height' with the value set $\{x \mid 0 \leq x \leq 100\}$. The attribute value 10 is *twice* the height of the value 5.

Scales usually are ordered in the sequence nominal, ordinal, interval and ratio, with the progression reflecting increasing information demands for scale definition. Hence promoting a variable implies the utilization of additional information or acceptance of a new assumption. Likewise, demotion of a variable involves relinquishing

some information. One approach to handling the problem of lack of homogeneity between attributes is to choose a particular scale type and then suitably transform the attributes to achieve homogeneity. There are several ways to accomplish this, depending on what kind of conversion is required. One of the most common methods to accomplish this is by *substitution*. As an example let us consider the attribute 'age' which is measured in years. This attribute has a scale of type ratio. To convert it to an ordinal attribute we can perform the following substitution:

- if $age < 10$, $age = child$
- if $20 > age \geq 10$, $age = teenager$
- if $age \geq 20$, $age = adult$.

What we have done is defined contiguous categories which are constructed by aggregating distinct attribute values. Thus, information loss takes three forms:-

1. distinction between objects in the same class are lost,
2. distinctions between objects in different classes retain order properties but magnitudes of their distinctions are lost or ignored,
3. there is no longer a meaningful reference point (in this case 0 years).

2.2.4 Similarity Measures

Similarity plays a fundamental role in theories of knowledge. It serves as the organizing principle in which individuals classify objects, form concepts and make generalizations. Most theoretical analysis of similarity are based on quantitative techniques. That is, objects are usually represented as points in some representation space and similarity is measured as the value of a distance function. Other quantitative techniques rely on either probabilistic or fuzzy set approaches. Recently, more

researchers have been considering qualitative similarity measures that are based on psychological principles of category theory. These quantitative and qualitative approaches to similarity are discussed in the next two sections, respectively.

Most cluster analysis methods require a measure of similarity to be defined for every pairwise combination of entities to be clustered. When clustering data objects the proximity of the individuals is usually expressed as a distance. It is the combined choices of attributes, transformations and similarity measures give operational meaning to the term 'natural association'.

2.2.5 Clustering Criteria

The term 'cluster' is often left undefined, but when it comes to finding clusters in real data, the term must bear a definite meaning. The choice of clustering criteria is tantamount to defining a cluster. It may not be possible to say just what a cluster is in abstract terms, but it can always be defined constructively through statement of criteria and implementing algorithm. As discussed below, clustering techniques are classified as either hierarchical or nonhierarchical depending on the method utilized to generate the clusters.

Hierarchical clustering

The similarity measures discussed briefly above may be used to construct similarity matrices describing the strength of all pairwise relationships among entities in the data set. The methods of the hierarchical cluster analysis operate on the similarity matrix to construct a tree depicting specified relationships among entities. The leaves of the tree each represent one entity while the root represents the entire collection of entities. Moving down the tree from the leaves toward the root depicts increasing aggregation of the entities into clusters. Hierarchical clustering methods which build a

tree from leaves to root are called agglomerative methods. Less common are divisive hierarchical methods which begin at the root and work toward the leaves.

Agglomerative clustering. Once a similarity matrix is defined the process of clustering data elements proceeds as follows:-

1. Begin with n clusters, each representing exactly one object in the data set.
2. Determine the highest similarity measure between a pair of clusters.
3. Reduce the number of clusters by one and join the two most similar clusters determined in step 2. Update similarity matrix after merger of clusters.
4. Repeat process until all the clusters are joined into one. This process will require $n - 1$ iterations.

Divisive Clustering. These methods are based on finding the groups which are the best separated from each other or most distinctive. The object is to split the data units (or objects) into groups as to minimize the value of some appropriate measure of similarity between two groups. The methods must consider $2^{m-1} - 1$ partitions of m data units into two groups. Thus, to avoid this computational complexity certain heuristics must be followed.

Nonhierarchical clustering methods

These methods are designed to cluster data units into a single classification of k clusters where k is either specified *a priori* or is determined as a part of the clustering method. The central idea in most of these methods is to choose some initial partition of the data units and then alter the cluster membership so as to obtain a better partition. These various algorithms which have been proposed differ as to what constitutes a 'better partition' and what methods may be used for achieving improvements.

Methods begin with an initial set of *clustering seeds* around which clusters may be formed. Thus, the choice of these seeds is critical to the clustering process. These seeds may be generated by:

1. taking the first k data units in the data set,
2. subjectively choosing any k data units,
3. randomly choosing k data objects by numbering the data objects from 1 to n and generating random numbers from 1 to n ,
4. An intuitively appealing goal is to choose seed points which span the data set, that is, to have most data points relatively close to the seed points while the seed points are themselves well separated from each other. One way of doing this is by taking the overall mean of the data objects as the first seed point (this might mean that the seed point is not an actual data object); select subsequent seed points by examining the data units in their input sequence and accept any data unit which is at least some specified distance from all previously chosen seed points; continue until all k seed points are chosen.

After the choice of clustering seeds, the data objects then become members of the closest clustering seed depending on the form of similarity measure used. The set of seed points are usually updated to reflect the members of the cluster. The most intuitive update procedure is that of the *nearest centroid sorting method* [1]. In this method the seed points are recomputed as the centroids of each of the clusters. The simplest iterative clustering method consists of altering these two processes until they converge to a stable configuration.

2.3 Quantitative Similarity Measures

Most conventional similarity measures are quantitative. In this section we will present the three most prominent numerical similarity measures. These measures are based on distance, probability and fuzzy sets, respectively. Objects can be represented as a point in an n -dimensional representation space and similarity is determined if two objects are within a certain threshold. Probabilistic approaches maximize the probability of two objects being similar by including them into the same category. Fuzzy set approaches utilize the notion of a fuzzy set which attempts to encode a concept by using a set membership function that acknowledges ambiguity. Data objects are considered members of a cluster if their *grades* of membership are within a certain threshold.

2.3.1 Distances

Most of the conventional clustering schemes represent data objects as points in an n -dimensional space. If a certain is within a certain threshold that point will be considered as a member of a certain cluster. Distance functions are classified as either metric or nonmetric. These two types of distance functions are discussed in the next two subsections.

Metric distance functions

Let S be an n -dimensional representational space where x, y and z are points in that space. That is, $x, y, z \in S$. Any metric distance function, D , must satisfy the following criteria:-

1. $D(x, y) = 0 \iff x = y$.
2. $D(x, y) \geq 0, \forall x, y \in S$.
3. $D(x, y) = D(y, x)$

$$4. D(x, y) \leq D(x, z) + D(z, y)$$

The first property implies that x is zero distance from itself and that any two points that are zero distance from each other are identical. The second point prohibits negative distances. The third point imposes symmetry by requiring that the distance from x to y be the same from y to x . The fourth property is known as the triangle property and states that the length of one side of the triangle is no longer than the sum of the other two sides. These properties are in accordance with intuitive notions because the popular concept of a distance is a Euclidean distance of elementary geometry, itself a metric.

A popular distance metric is the Minkowski metric. Then the Minkowski metric between two data objects is

$$D_p(x, y) = \left[\sum_{i=1}^{i=n} |x_i - y_i|^p \right]^{\frac{1}{p}}$$

where x_i refers to the i th attribute value of the data object. By choosing different values of p many different metric distance functions can be obtained. The familiar Euclidean distance is obtained by taking $p = 2$.

$$D_2(x, y) = \left[\sum_{i=1}^{i=n} |x_i - y_i|^2 \right]^{\frac{1}{2}}$$

Nonmetric distance functions

Nonmetric distance functions are less common than metric functions. Yet, some researchers have developed certain nonmetric function for specific research requirements. Consider Lance and Williams (1966) suggest the nonmetric distance function:-

$$D_{LW}(x, y) = \frac{\sum_{i=1}^{i=n} |x_i - y_i|}{\sum_{i=1}^{i=n} (x_i + y_i)}$$

as a distance between data units x and y . Notice that the numerator of the function

is simply the Minkowski metric with $p = 1$; while the denominator is viewed as a measure of the gross magnitude of the two data units. This is not a metric because the values could be negative and because it might not satisfy the triangle rule.

Notice that the units of measurements of the attribute values play a key role in the magnitude of the distance function. This refers to both metric and nonmetric functions. The choice between grams and kilograms in one attribute scale and meters and kilometers in another will affect the outcome. Thus, there is an implicit weighting of attributes. Moreover, the scales of different attributes are combined to achieve a single measure of distance. But, how can this measure be correctly interpreted? One suggested solution is to *equalize* the variables in some appropriate manner. As mentioned by Anderberg [1], the principal idea in equilization is the removal of the artifact of the measurement unit and anchoring each variable to some common numerical scale.

Disposing of the measurement unit involves dividing all the scores of the variable by an appropriate equalizing factor expressed in the same units. The sense in which the variable is equalized depends on which kind of equalizing factor is chosen. A popular equalizing factor for ratio variables is that of the mean. The expected value on a variable equalized by the mean is one and the expected difference between values of the same variable for two different objects is zero.

2.3.2 Probabilistic Approaches

Probabilistic approaches generate clusters by maximizing certain *probabilistic* clustering criteria. These criteria vary from approach to approach. Typically, inductive inference in these approaches takes the form of generating information by which examples of concepts are identified, in other words classification rules.

ID3 [45] is a comparatively simple mechanism that utilizes this approach for discovering a classification rule for a collection of objects belonging to two classes. ID3 was specifically designed to handle masses of objects and in fact its computation only increases linearly with difficulty when modeled as a product of the number of data objects considered, the number of attributes used to describe these data objects and the number of concepts to be developed (measured by the number of nodes in the decision tree). On the negative side this linearity is purchased at the cost of descriptive power.

The concepts developed by ID3 can only take the form of decision trees based on the attributes given and this language is much more restrictive than first-order logic or multi-valued logic used in other systems. Each object must be described in terms of a fixed set of attributes, each of which has its own set of possible attribute values. A classification rule in the form of a decision tree can be constructed for any such collection C of objects. If C is empty, then we associate it arbitrarily with either class. If all objects in C belong to the same class, then the decision tree is a leaf bearing the class name. Otherwise, C contains representatives of both classes. We select an attribute and partition C into disjoint sets C_1, \dots, C_n , where C_i contains those members of C that have the i th value of the selected attribute. Each of these subcollections is handled by the same rule-forming strategy. The eventual outcome is a tree in which each leaf carries a class name, each interior node specifies an attribute to be tested, with a branch corresponding to each possible value of that attribute.

An object is classified by starting at the root of the decision tree, finding the value of the tested attribute in the given object, taking the branch appropriate to that value and continuing in this fashion until the leaf is reached. Notice that classifying a particular object may involve evaluating only a small number of attributes depending

on the length of the path from the root of the tree to the appropriate leaf. This rule-forming procedure will always work provided there are no two objects belonging to different classes but having identical values for each attribute; in such a case the attributes are inadequate for the classification task. The whole skill in this style of induction lies in selecting a useful attribute to test for a given collection of objects so that the final tree is minimal in some sense, i.e. to minimize the path from the root of the tree to the leaves. ID3 uses an information-theoretic approach aimed at minimizing the expected number of tests to classify an object.

A decision tree may be regarded as an information source that, given an object, generates a message which is the class of that object. The attribute selection part of ID3 is based on the plausible assumption that the complexity of the decision tree is strongly related to the amount of information conveyed by the message. If the probability of these messages are p_1 and p_2 , respectively, the expected information content is

$$-p_1 \log(p_1) - p_2 \log(p_2)$$

with a known set C of objects we can approximate these probabilities by relative frequencies of the occurrence.

Let $M(C)$ be the expected information content of a message from a decision tree for a set C of objects. Moreover, $M(\{\}) = 0$. The new expected information content for a certain attribute A is

$$B(C, A) = (\text{probability that the value of } A \text{ is } A_i) \times M(C_i)$$

where again we can replace the probabilities by relative frequencies. The suggested choice of attributes to test next is that which gains the most information, in other words, for which

<u>Class 1</u>	<u>Class 2</u>
(tall, blond, brown)	(short, blond, blue)
(short, dark, blue)	(tall, red, blue)
(tall, dark, blue)	(tall, blond, blue)
(tall, dark, brown)	
(short, blond, brown)	

Figure 2.1. The set of data objects.

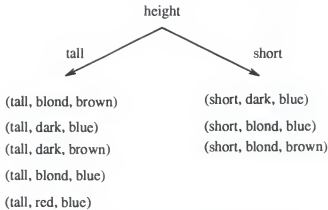


Figure 2.2. Binary attribute discrimination based on 'height'.

$$M(C) - B(C, A)$$

is a maximal. In essence we are trying to find the most discriminating attributes.

To illustrate this idea, consider the collection of objects given in Figure 2.1. Each object is described in terms of three attributes: 'height' with values $\{tall, short\}$, 'hair' with values $\{dark, red, blond\}$ and 'eyes' with values $\{blue, brown\}$.

The collection of objects in Figure 2.1 contains 5 in class 1 and 3 in class 2, so

$$M(C) = -\frac{3}{8} \log_2 \frac{3}{8} - \frac{5}{8} \log_2 \frac{5}{8}$$

Testing the first attribute gives the result shown in Figure 2.2.

The information still needed for a rule for the 'tall' branch is

$$-\frac{2}{5} \log_2 \frac{2}{5} - \frac{3}{5} \log_2 \frac{3}{5} = 0.971$$

and for the short branch

$$-\frac{1}{3} \log_2 \frac{1}{3} - \frac{2}{3} \log_2 \frac{2}{3} = 0.918$$

Thus the expected information content

$$B(C, height) = \frac{5}{8} * 0.971 + \frac{3}{8} * 0.918 = 0.951$$

The information gained by testing this attribute is

$$0.954 - 0.951 = 0.003 \text{ bits}$$

which is negligible.

By repeating this process for the second attribute, we find that the information gained by testing ‘eyes’ comes to 0.347. The information gained by testing the attribute ‘hair’ is 0.454. Thus the principle of maximizing expected information gain would lead ID3 to select ‘hair’ as the attribute to form the root of the decision tree. By continuing this process, we are able to generate the decision shown in Figure 2.3.

2.3.3 Fuzzy Sets

In real life there are seldom such clear-cut boundaries to clusters. In Zadeh’s words,

More often than not, the classes of objects encountered in the real world do not have precisely defined criteria of membership. For example, the class of animals clearly includes dogs, horses, birds, etc as its members, and clearly excludes such objects as rocks, fluids, plants, etc. However, such objects as starfish, bacteria, etc. have an ambiguous status with respect to the class of animals. ([23], p. 361.)

To model this uncertainty, Zadeh introduced the concept of a fuzzy set. In classical set theory, an object is either a member of a set or not. Fuzzy set theory tries to encode a concept of set membership that acknowledges ambiguity. Consider a classical set A . Associated with it is a membership function, $u_A()$, defined by

$$u_A(x) = \begin{cases} 1 & x \in A \\ 0 & x \notin A \end{cases}$$

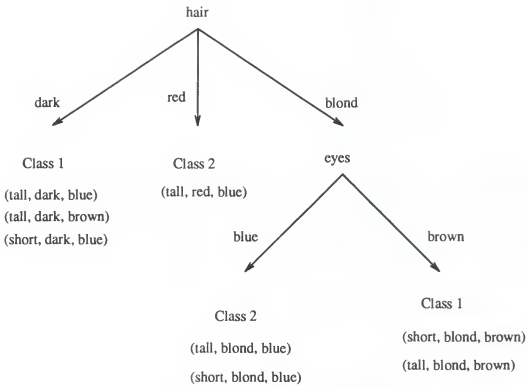


Figure 2.3. Final decision tree.

Thus, $u_A()$ may take one of precisely two values, 1 or 0. If we prepare to allow ambiguity we would allow $u_A()$ to take any value between 0 and 1. The membership function assigns values, in the range 0 to 1, to each element of the set. This value indicates the *grade* of membership in the set. The greater the value the greater the degree of membership in the set. For example, we can define a fuzzy set *YOUNG* (denoting youth) as a set of ages 20, 30 and 40 with grades 0.9, 0.7, and 0.3, respectively.

In classical set theory, the intersection of two sets is defined by

$$A \cap B = \{x | x \in A \text{ and } x \in B\}$$

This corresponds to a set membership function,

$$u_{A \cap B}(x) = \begin{cases} 1 & x \in A \text{ and } x \in B \\ 0 & \text{otherwise} \end{cases}$$

A little thought shows that arithmetically,

$$u_{A \cap B}(x) = \min\{u_A(x), u_B(x)\}$$

In his original paper Zadeh suggested the above definition to be used for the intersection of two fuzzy sets, although he readily admitted that there were other possibilities, e.g. multiplication. Zadeh also suggested that the union and complement might be defined as

$$u_{A \cup B}(x) = \max\{u_A(x), u_B(x)\}$$

$$u_{\bar{A}}(x) = 1 - u_A(x)$$

Thus, classical (non-fuzzy) set theory models an idealized person's handling of classes of objects within his perception. It assumes that he perceives objects as either fully possessing the properties that define a class or not possessing them at all. Fuzzy set theory attempts to extend this model by allowing him to perceive properties that are possessed to varying degrees.

The other major type of imprecise data is fuzzy data. The basic concept underlying fuzzy data is that of a fuzzy set. A fuzzy set A is a set of elements with an associated set membership function $u_A()$.

The FRDB system [52] is an experimental database system based on the ideas mentioned above. To manipulate fuzzy databases, the standard relational algebra must be extended to fuzzy relations. Specifically, the imprecision in the values of the tuples creates problems of value identification (e.g. in the join or in the removal of duplicates in the projections). Also, standard mathematical comparators such as $=$ or $<$ are superseded by fuzzy set comparators such as *similar-to* or *much-greater-than*. It is this type of fuzzy operators that allow the expression of fuzzy retrieval requests. Each query in FRDB is a sequence of statements that perform fuzzy retrieval operations on the database. An optional argument of each statement is a threshold value, between 0 and 1, which is used to filter the tuples of the result by selecting only those whose grade values exceed that of the threshold. For example, in FRDB a user might specify a query ‘select person, whose age is *YOUNG*, with threshold value of 0.8’. Here, *YOUNG* is a fuzzy set of values, person is a relation with attribute age, the domain of which is *YOUNG*. Thus, the user wants all persons whose ages have grades greater than 0.8.

2.4 Qualitative Similarity Measures

The grouping of objects into classes is an issue that concerns many diverse disciplines such as philosophy, psychology, artificial intelligence and in addition to data modeling. It has long been argued that human categorization is not an ad hoc process but rather a process that is guided by psychological principles of categorization [47]. The aim of a theory of categorization is to explore these principles. In the next

two subsections we will highlight concepts of category theory that are relevant to the purpose at hand and the applications of these principles, respectively.

2.4.1 A Theory Of Categorization

Categories are *groups of objects which are considered similar in a certain context* [47]. To categorize an instance means to consider it, for purposes of that categorization, not only similar to other instances in the same category but different from instances not in that category. Category boundaries try to reflect the natural discontinuities that occur in perceptual and functional features of our world. Tversky [51] applies a set-theoretic approach to similarity in which objects are represented as collections of features and similarity is described as a feature-matching process. Let $s(a, b)$ be a measure of similarity between two objects a and b . Thus,

$$s(a, b) = F(a \cap b, a - b, b - a).$$

The similarity of a and b is expressed as a function F of three arguments: $a \cap b$, the features that are common to both a and b ; $a - b$, the features that belong to a and not to b ; $b - a$, the features that belong to b but not to a . Moreover,

$$s(a, b) \geq s(a, c)$$

whenever

$$a \cap c \subseteq a \cap b, \quad a - c \subseteq a - b, \quad \text{and} \quad c - a \subseteq b - a.$$

That is, the similarity increases with the addition of the common features and/or the deletion of the distinctive features.

A greater overlap of instance features *does not* necessarily imply higher probability of membership in the same category. The cohesion between members of a category does not lie in the overall degree of feature overlap, but in the overlapping of relevant, *context-dependent features* [25, 28, 47, 50]. Context affects both the level of abstraction at which an instance is considered and the relevance of its features.

Consider batteries and reservoirs. They do not have very much in common, but in the context of ‘energy storing devices’, they are very similar. They are even considered analogous, as in ‘batteries are reservoirs’ [25]. By the same token, humans around the world share many common attributes but in the context of ‘religious beliefs’, are divided into many categories. Simply put, categories are viewed as complex conceptual clusters rather than simple predicates.

Principles of category formation

Two general principles guide the formation of categories. The first, *cognitive economy*, deals with the functional aspect of a category system to provide maximum information with the least cognitive effort. The second principle deals with the structure of the perceived world and asserts that objects of the world are perceived to possess high correlational structure. That is, the world is not an unstructured total set of equiprobable co-occurring attributes. Thus, maximum information with the least cognitive effort is achieved if categories map the perceived world structure as closely and truthfully as possible. To reiterate, category boundaries try to reflect the natural discontinuities that occur in the perceptual and functional features of our world.

The category system

A category system is a taxonomy by which categories are related together through class inclusion. Such a system has a vertical and horizontal dimension. The vertical dimension reflects the level of abstraction or *inclusiveness* of a category, similar to that of a generalization hierarchy. As a category becomes more abstract, and therefore more inclusive (contains more instances), there would be very little that all the instances have in common. This is termed as the *family resemblance* effect

[47]. What do all ‘chairs’ have in common? Does a bean-bag chair satisfy this description? This is another indication why necessary and sufficient conditions for class membership are not adequate. The implication of the two principles of categorization on the vertical dimension is that not all possible levels of categorization are equally good or useful. Rather, the *basic level* of abstraction is the level which maximizes the similarity between category members in a certain context [20, 47]. Consider the following category system: electronic equipment, computer and personal computer. When referring to a Macintosh, an inventory clerk may identify it as electronic equipment while an end-user will probably relate to its computing capabilities and identify it as a personal computer.

The horizontal dimension concerns the segmentation of categories at the same level of abstraction. Most, if not all, categories do not have clear-cut boundaries. To argue that categories reflect the attribute structure of their objects does not imply that such attribute clusters are necessarily discontinuous. The implication of the two principles of categorization for the horizontal dimension is to increase the distinctiveness of the categories. To achieve this separateness and clarity, categories tend to become defined in terms of prototypes or prototypical instances that contain the features most representative of the instances inside the category and least representative of instances outside the category. That is, there exists a hypothetical prototypical instance that represents typical members of a category. Prototypes appear to be just those members of a category that reflect the redundancy structure of a category as a whole. For example, to most people a chair with four legs, a back and a seat is more typical than a chair with three legs and no back. Most data models ignore prototypes and treat all instances as equal members of the class. The notion of prototypes is closely related to that of default values which are typical attribute-value pairs of instances in a class. It is not required that all instances of the class have these default

values. But, in the absence of specified attribute values, default reasoning may be used during query processing or updating to predict these missing values.

2.4.2 Conceptual Clustering Techniques

Conceptual clustering is a technique based on category theory that creates classes of objects that are conceptually cohesive [49]. This cohesiveness depends not only on the properties of the individual objects, but on other objects and the context in which the class is formed. In contrast, classes in conventional data analysis, based on numerical taxonomy and regression analysis [35], are formulated solely on the basis of a numerical measure of similarity between the objects. The similarity between objects is determined as the value of a mathematical function applied to each object's description. If this value was within a certain threshold, the object would be considered a member of that class. This type of similarity measure is context-free; it is not able to capture the properties of a class that are not derivable from the properties of the individual objects [35]. For instance, by using this type of similarity measure we could categorize objects having an attribute 'location' as being similar (that is, in the vicinity of each other) if their values for 'location' were within a certain range. But, it is doubtful whether this range will be the same in the context of 'countries in the vicinity' and 'people in the vicinity'.

Conceptual clustering is an inductive process. The goal of induction is to formulate a general assertion that implies the given object descriptions. In contrast to deduction, the starting premise of induction are specific facts rather than axioms. For any given set of facts, there will be a potentially large number of hypotheses that could be generated that imply these facts. Background knowledge is used to constrain this process. Components of *background knowledge* include information about object

properties, a variety of inference rules (deductive and inductive), heuristics and specialized procedures that allow the system to generate logical consequences of given assertions and new descriptions [50]. The context determines not only the relevant attributes but also the level of abstraction (basic level). For example, doctors have many attributes and can be classified in many different ways. But, when trying to find a competent doctor, hair color or height are usually not relevant to your choice even though they are valid classifications. Relevancy of attributes cannot usually be directly determined; therefore, deductive rules, heuristics and specialized routines are required. Deductive rules and heuristics will capture the interrelationships between attributes and therefore determine attribute relevancy. Specialized routines are required to transform the domain of the attribute from one representational space to another to facilitate comparisons and other operations. In a nutshell, conceptual clustering can be viewed as *a heuristic search through the space of possible symbolic descriptions generated by the application of various inference rules to object attributes and constrained by background knowledge* [35, 50].

According to Michaliski [35, 36, 50], the general paradigm of conceptual clustering is as follows. Given 1) a set of object descriptions (in our case these are attribute-value vectors), 2) a tentative inductive assertion, which may be null, and 3) background knowledge to guide the clustering process, generate a class taxonomy that implies the object descriptions and satisfies the background knowledge.

Previous work in conceptual clustering

CLUSTER/2 is a conceptual clustering technique developed by Michalski and Stepp [36]. It utilizes Michalski's INDUCE/2 algorithm to generate class descriptions from a set of instances. Although INDUCE/2 was intended to be used as a form of learning from example, it was applied in such a way that CLUSTER/2 would operate

without a tutor providing examples of class members. CLUSTER/2 utilizes a set of rules incorporated in the Lexical Evaluation Function (LEF) to determine category quality. Classes that do not meet the quality control are rejected. CLUSTER/S [50], based on the work in CLUSTER/2, utilizes a goal-dependency network (GDN) to derive attributes related to the initial goal. For example, if the goal is 'Survive', then the concept 'Eat Food' is determined to be relevant through the GDN. Classes are formed that correspond to relevant attributes identified by the GDN.

COBWEB [20] is a conceptual clustering technique that utilizes a probabilistic approach to the organization of data. It does this by capturing attribute inter-correlations as classification tree nodes and generating inferences as a by-product of classification. A statistical measure, the *class utility measure*, is used to evaluate the quality of different classes (clustering patterns) over the same set of instances. This class utility measure is designed to maximize the ability to predict instance attributes from knowledge of class membership. That is, the utility favors classes that maximize the amount of information that can be inferred from knowledge of class membership. Category utility rewards information rich categories and is thus of generic value, but it can also be viewed as a trade-off of intra-class object similarity and inter-class dissimilarity. Objects clustered by COBWEB are described in terms of attribute-value pairs. For an attribute-value pair $A_i = V_{ij}$, and class C_k , intra-class similarity is measured in terms of the conditional probability $P(A_i = V_{ij}|C_k)$. The larger this probability, the greater the proportion of class members sharing the same value (V_{ij}) and thus the more *predictable* the value is of class members.

Inter-class similarity is measured in terms of $P(C_k|A_i = V_{ij})$. The larger this probability, the fewer objects in contrasting classes that share this value, and thus the more *predictive* the value is of the class.

Attribute value predictability and predictiveness are combined into a measure of partition quality, specifically

$$\sum_{k=1}^{k=n} \sum_i P(A_i = V_{ij}) P(C_k | A_i = V_{ij}) P(A_i = V_{ij} | C_k)$$

, is a tradeoff in predictability and predictiveness that is summed for all classes (k), attributes (i) and values (j).

Notice that the predictability $P(A_i = V_{ij})$ weighs the importance of individual values, in essence saying that it is more important to increase class conditional predictability and predictiveness of frequently occurring values than infrequently occurring ones. This function can also be regarded as rewarding the inference potential of object class partitions. More precisely, if we use Bayes rule,

$$P(A_i = V_{ij}) P(C_k | A_i = V_{ij}) = P(C_k) P(A_i = V_{ij} | C_k)$$

so by substitution the above function equals

$$\sum_{k=1}^{k=n} P(C_k) \sum_i \sum_j P(A_i = V_{ij} | C_k)^2.$$

where $\sum_i \sum_j P(A_i = V_{ij} | C_k)^2$ is the expected attribute value that can be correctly guessed for any arbitrary member of class C_k .

Category utility can be computed given $P(C_k)$ is known for each category of a partition, as is $P(A_i = V_{ij} | C_k)$ for all attribute values. Such a category representation is termed a probabilistic concept. This requires the storage of the probabilities.

COBWEB incrementally incorporates objects into a classification hierarchy. Given an initially *empty* hierarchy, over an incrementally presented series of objects, a hierarchical classification is formed, where each node is a probabilistic concept representing each class. The incorporation of an object is basically a process of classifying the object in descending the tree along an appropriate path, updating statistical information along the way.

ACME (Analogical Constraint Mapping Engine) [28] is a program that utilizes a connectionist approach to conceptual clustering. ACME applies an analogical mapping between source and target analogs based upon interacting structural, semantic and pragmatic considerations. The *structural constraint* encourages mappings that maximize the consistency of the relational correspondences between elements of the two analogs. The *semantic similarity constraint* supports mapping hypotheses to the degree that mapped predicates have similar meaning. The constraint *pragmatic similarity* favors mappings involving elements that the user believes to be important in order to achieve the purpose for which the analogy is being used.

In this program constraints of a class are represented by means of a network of supporting and competing hypotheses regarding which elements of the instance to map. A cooperative algorithm for parallel constraint satisfaction identifies mapping hypotheses that collectively represent the overall mapping that best fits the interacting constraints. A cooperative algorithm is a procedure for parallel satisfaction of a set of competing constraints. A network of nodes is established in which each node represents a possible pair of matched points; excitatory and inhibitory connections between nodes represent constraints. The network is then allowed to run in order to find a globally optimal set of match hypotheses. The network incorporated by ACME utilizes numerical weights, thus a class modeled by this network is determined by a numerical measure of similarity.

CHAPTER 3

SCHEMA INTEGRATION THROUGH CONCEPTUAL CLUSTERING

In recent years there has been a substantial amount of research directed towards schema integration. The main reasons for this are twofold. There has been an increase in the available commercial and experimental database management systems which are implemented with different incompatible data models. Second, the demand for complex databases with numerous users necessitates the integration of several schema designed specifically to meet each user groups needs. Schema integration [6] is the activity of integrating schemas of proposed or existing databases into a global, unified schema. This type of integration process encompasses view integration and database integration. View integration [43] produces a global conceptual schema of a proposed database by integrating views that model the users' data requirements and the application processing requirements. This is usually done at the conceptual schema design stage. Database integration, in the distributed or federated context, produces a virtual view of all the databases taken together. It is this type of integration that we are concerned with in this chapter.

3.1 Conventional Schema Integration Methodologies

The basic problems to be dealt with during schema integration stem from structural and semantic diversities of the schemas to be merged [6]. As discussed by Batini et al [6], any conventional schema integration methodology can be considered as a mixture of these following activities:

1. **Pre-integration.** This step entails an analysis of the schemas to establish the integration policy. This process will determine the schemas to be integrated, the

order of integration, and assignment preferences to entire schemas or portions thereof.

2. **Comparison of Schema.** In this step schemas are studied and compared to determine possible correspondences or conflicts. Essentially, this step will determine the interschema properties.
3. **Conforming the Schema.** After conflicts are detected, an attempt is made to resolve them to facilitate the merging of the schemas. This process entails transforming schemas into compatible or aligned schemas.
4. **Merging and Restructuring.** At this stage the schemas are ready to be superimposed and, depending on the integration algorithm, several intermediate integrated schemas could be generated. These intermediate schemas could be restructured to attain some desirable qualities. The resultant global schema should be complete and correct (must represent all the underlying schemas), minimal (the same concept cannot be represented more than once), and understandable (easily understood by the designer and the end users).

MULTIBASE [48] is a software systems for integrating access to pre-existing heterogeneous database systems. This system provides the user with a unified global schema and a single high-level query language. Prior to integration, all the local host schemas (LHS) are converted into a common data model, the Functional Data model. These normalized schema are the local schema (LS). Local schemas are in turn merged together into a global schema by a mapping language facility. This facility utilizes an *integration database* that contains information needed for integration such as the mappings between different scales, statistical information about imprecise data and other information needed for reconciling inconsistencies between databases.

Since local host schema can be defined in the relational, CODASYL or file model, how the LHS is mapped into the LS depends on the different constructs of these data models. In a similar vein, Motro and Buneman [42] has suggested a method for integrating databases into a conceptual ‘superview’ through a set of transformations. Each transformation defines a mapping of queries against the superview into the appropriate set of queries against the underlying databases.

Elmasri et al. [17] proposed several algorithms for the integration of federated databases. These algorithms assume that the local schemas are translated into a common data model, the E-C-R data model. The pre-integration phase involves schema analysis and modification to *align* the schemas. Integration then proceeds by integrating compatible or equivalent constructs. Correspondence between constructs of the schema are specified as assertions. Attribute assertions are used to order pairs of object classes based on the number of equivalent attributes they share. Whenever two attributes are equivalent, an attribute equivalence assertion is used to express that fact. These assertions are summarized in the attribute class similarity (ACS) matrix. An entry in the ACS matrix is either 1 if the corresponding attribute is a member of the corresponding entity set, and 0 otherwise. The object class similarity (OCS) matrix is calculated from the ACS matrix by the following equation:

$$OCS_{ij} = \sum_{k=1}^m (AND(ACS_{ik}, ACS_{jk}))$$

where AND is the logical AND. If $OCS_{ij} > OCS_{ik}$ then O_i and O_j have more equivalent attributes than O_i and O_k . Thus, O_i and O_j are more likely to be integrated.

SIS (Schema Integration System) [14] is an expert system that assists users in finding the overlap, determining whether conflicts exist, modifying the schema and in defining mappings between constructs of the two schema. Complete semantic or syntactic equivalence between constructs of databases being integrated are rare in real

world problems. Hence the SIS system attempts to quantify the similarity between a pair of constructs in such a way that the integration is presented with lists of pairs of constructs ordered by their measure of similarity. To accomplish this, SIS utilizes a *uni-dimensional syntactic comparison function*. Let A be the set of elements of the first schema and B be the set of elements of the second schema. A function F is defined such that

$$F : A \times B \rightarrow [0, 1]$$

The function F will have the value 1 if two constructs have the same name, 0 if they are completely different and some intermediate value if they differ by a minor spelling discrepancy. These intermediate values represent the resemblance between the constructs. Hence, F can be viewed as a membership function of a fuzzy subset of $A \times B$.

The main drawback of current schema integration methodologies is that they are *data model dependent* and integrate at the *construct level*. All of the four steps outlined [6] entail some kind of analysis of schemas. That is, the main emphasis is on integrating constructs from a particular data model (schemas must be ‘normalized’ by translation into a common data model prior to integration). These methodologies reason at the class level *to the exclusion of instances*. Furthermore, the integration process is marred by extensive designer interaction. No integration tool is totally automated (e.g. see the ten methodologies surveyed by Batini et al. [6] or the most automated of the current methodologies [11]). Among other causes, this could be attributed to the ad hoc and informal treatment of classes in semantic data models. Without a formal treatment of classes, the designer must use his intuition and experience to interpret the *meaning* of each class. Moreover, current integration tools depend on simple measures of similarity such as string and domain comparisons [6].

These similarity measures are incapable of capturing essential interrelationships between database objects vital to the integration process. As an example, consider a personnel database and a production database both having the attribute 'position'. These attributes would be considered similar by string comparison even though in the personnel database 'position' might represent the rank of an employee while in the production database it might represent the location of the product. This is referred to as a 'homonym' and must be detected by the user to avoid erroneous integration. Conversely, the attributes 'birthdate' and 'age' would not be considered similar by these measures of similarity even though one can be derived from the other ($age = date - birth-date$).

By the application of conceptual clustering to database integration, we are able to automate the process significantly and generate a more descriptive schema. As mentioned earlier, the conceptual clustering technique generates conceptually cohesive classes that are based on instance descriptions. In applying this technique, we reason over the actual instances of the databases. Schema generation occurs as a result of conceptually clustering the underlying data instances of different (possibly heterogeneous) databases and guiding this process by specifying a clustering seed. Therefore, schemas generated by this method are more precise as they reflect the actual instances stored in the database and the context specified by the user more accurately. The data model used to represent these schemas is of *no consequence*. Any semantic data model that supports classes, relationships between classes, an IS-A hierarchy and structural cardinality constraints can be used. Therefore, no analysis is required to compare the different constructs of the data models, conform schemas, or merge and restructure schemas as done in the conventional schema integration methodologies. This process can be totally automated. However, an analysis of the interrelationships between the attributes of the instances is required. Such relationships as set-subset,

synonymity, logical and mathematical relationships should be determined. It is this information that guides the process and the quality of the classes generated are dependent on such information. Notice that schemas generated by this approach, since they are represented in conventional semantic data models, might entail necessary and sufficient conditions. This is not a contradiction, as this approach should be complemented by schema evolution and exception handling techniques, such as those specified in [7, 10, 34], to allow the schema to evolve and to continue to represent the application domain more accurately.

The remainder of this chapter is organized as follows. Section 3.2 presents the system architecture and the data model which we are presently using to validate our approach. Section 3.3 illustrates the view generation algorithm with a detailed example. Implementation issues are discussed in Section 3.4.

3.2 Architecture of a Conceptual Clustering System

In this section we describe the Conceptual Clustering System (CCS) in detail. The system architecture will first be introduced. To illustrate the views generated by the system the CANDIDE semantic data model will then be briefly discussed. This is for illustrative purposes only as *the data model is of no consequence*. To reiterate, any semantic data model that supports classes, relationships between classes, an IS-A hierarchy and structural cardinality constraints, such as extended entity relationship models (e.g. the ECR model [19], or several other models surveyed [29]) could be used. In section 3.3 the clustering algorithm will be discussed in detail in conjunction with an example.

3.2.1 System Architecture

CCS consists of four main modules, as shown in Fig. 3.1. The four modules are the databases (to which it is connected), the background knowledge base, view

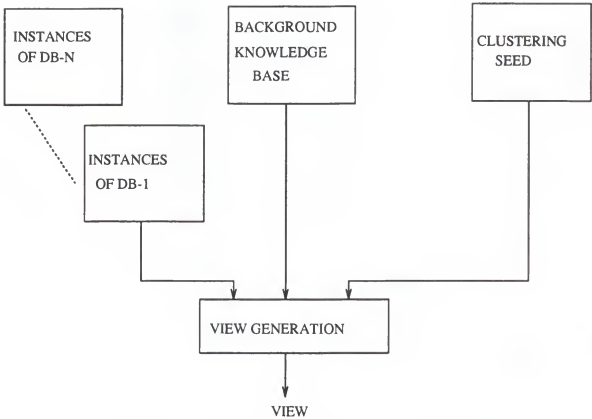


Figure 3.1. Conceptual Clustering System Architecture

specification (clustering seed) and the view generator. The following subsections will describe these four components.

Databases

Several databases are connected to the CCS. We assume here a federated environment. Only actual instances are considered. These instances are assumed to be in attribute-value pairs of possibly variable sizes. We also assume here that the data formats of the instance descriptions are uniform. While uniform representation of instances could easily be accomplished through reformatting routines, the correlation

between database instances in different databases seems to require designer intervention [6]. We, therefore, assume that these databases are *disjoint* in the sense that they model different, mutually exclusive, domains.

Background knowledge base

The background knowledge of the system is stored in this module. This knowledge base is a set of facts that models specific interrelationships between attributes. These interrelationships allow the system to deduce relevant attributes with respect to the clustering seed specified by the user. To represent these interrelationships we use a combination of two models: a semantic network and a dataflow graph. An alternative to this approach would have been the use of first-order predicate logic. However, such a solution would not be suitable due to the computational complexity of the associated general theorem prover. As noted [12], a trade-off exists between the expressivity of a knowledge representation scheme and the complexity of its reasoning procedures. Consequently, we chose a representation scheme that is less expressive, yet powerful enough to be useful.

We use a specific kind of semantic network to model set-subset, synonymy and logical relationships between attributes. Formally, our semantic network is defined as a double $\{V, E\}$, where V stands for the set of nodes and E stands for the set of arcs. The nodes in V include the attributes known to the system plus special nodes described below. The set of arcs E denote the relationships among attributes. For each element of F of E , there exists a mapping:

$$F : V \times V \longrightarrow \{TRUE, FALSE\}.$$

$F(n_j, n_k)$ is *TRUE* if there exists an arc of class F between n_j and n_k , and *FALSE* otherwise. V is partitioned into $\{n_{oid}, V_{attr}\}$ where n_{oid} represents the node corresponding to the set of object identifications (OIDs) of instances and V_{attr} represents

the set of nodes corresponding to the attributes known to the system. The elements of these different types of nodes are connected by a set of arcs, E , which is partitioned into $\{subset, synonym, logical\}$. These arcs model the following interrelationships among attributes:

1. **Set-subset** relationship, denoted by the arc $subset(attr_j, attr_k)$ specifies that the domain of $attr_j$ is a subset of that of $attr_k$. That is,

$$dom(attr_j) \subset dom(attr_k)$$

For example, $subset(chairman, advises)$ represents the fact that all those who chair a committee must also advise. In other words, the domain of ‘chairman’ is a subset of that of ‘advises’. Another example is that of $subset(teaches, n_{oid})$. This denotes that the domain of attribute ‘teaches’ is a set of object identifiers (OIDs) and is therefore a subset of n_{oid} .

2. **Synonymity**, denoted by the arc $synonym(attr_j, attr_k)$ specifies that $attr_j$ is a synonym of $attr_k$. That is, it is considered equivalent. For example, $synonym(course-taught, teaches)$ represents the fact that the attribute ‘course-taught’ is a synonym of, and therefore considered equivalent to, ‘teaches’. Notice that only the nodes of the set V_{attr} can be joined by such arcs, i.e.

$$attr_j, attr_k \in V_{attr}$$

That is, n_{oid} cannot participate in such relationships because there is no other attribute that will exhibit the property of object identifiers.

3. **Logical** implication, denoted by the arc $logical(attr_j, attr_k)$, specifies that $attr_j$ is a logical consequence of $attr_k$. For example, $logical(teaches, advises)$, specifies that all who advise must also teach. Also, in this case n_{oid} cannot be connected by such arcs.

All the above constructs exhibit the transitivity property. That is,

$$\begin{aligned} \text{subset}(\text{attr}_1, \text{attr}_2) \wedge \text{subset}(\text{attr}_2, \text{attr}_3) &\implies \text{subset}(\text{attr}_1, \text{attr}_3) \\ \text{synonym}(\text{attr}_1, \text{attr}_2) \wedge \text{synonym}(\text{attr}_2, \text{attr}_3) &\implies \text{synonym}(\text{attr}_1, \text{attr}_3) \\ \text{logical}(\text{attr}_1, \text{attr}_2) \wedge \text{logical}(\text{attr}_2, \text{attr}_3) &\implies \text{logical}(\text{attr}_1, \text{attr}_3) \end{aligned}$$

While the *subset* and *logical* constructs are antisymmetric, the *synonym* construct is symmetric. That is,

$$\text{synonym}(\text{attr}_1, \text{attr}_2) \iff \text{synonym}(\text{attr}_2, \text{attr}_1).$$

Through the utilization of these properties the system is capable of inferring facts not explicitly specified. An example semantic network in Fig. 3.2a shows some of the interrelationships between the attributes mentioned above. Note that the synonym interrelationship is shown to be bidirectional.

To model mathematical relationships between the attributes a functional (acyclic) dataflow graph is used. These mathematical relationships permit the system to transform the domain of an attribute from one representational space to another. Formally, this functional dataflow graph is a triple $\{V_{\text{math-attr}}, V_{\text{actor}}, E_{\text{actor}}\}$, where $V_{\text{math-attr}}$ is a set of nodes corresponding to the attributes participating in such mathematical relationships.

$$V_{\text{math-attr}} \subseteq V_{\text{attr}},$$

because n_{oid} cannot participate in such relationships. V_{actor} is a set of nodes corresponding to *actors*. These actors perform mathematical operations on their inputs. V_{actor} is partitioned into $\{plus, minus, mult, div\}$ corresponding to the mathematical operations of addition, subtraction, multiplication and division, respectively. This set can be easily expanded to include other mathematical operations. E_{actor} is a set of directed arcs that connect the different types of nodes together and determine the

input and output of each actor node. Every actor node must be attached by at least one arc [49].

A dataflow graph that represents the following mathematical relationship

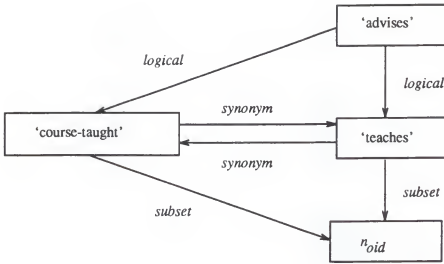
$$\text{math}(\text{salary} = \text{hrs-worked} \times \text{wages-per-hr})$$

is shown in Fig. 3.2b where attribute nodes are represented by boxes, actor nodes by octagons and arcs by dashed arrows.

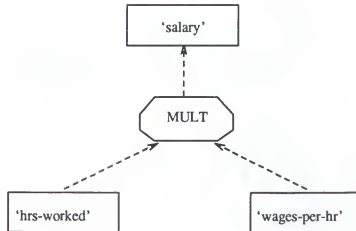
Due to the structural and semantic diversities of the attributes to be dealt with, several *conflicts* may arise. These conflicts can be characterized as naming conflicts and structural conflicts. Naming conflicts arise because people from different application areas refer to data using their own terminology. This gives rise to *homonyms* and *synonyms*. A homonym occurs when attributes, referenced by the same name, refer to different concepts. For example, 'reference-number' may refer to a case number in a legal database and to a part number in a suppliers database. As proposed in [6] we can alleviate this problem by using a full naming convention by concatenating the name of the database to that of the attribute, that is the attribute is referenced by

database-name . attribute-name.

A synonym occurs when the same concept is described by more than one name. As mentioned earlier, this relationship is captured by the *synonym* construct in the background knowledge base. Homonyms can be detected by comparing concepts with the same name in different databases, while synonyms can only be detected after external specification by the designer. Structural conflicts arise as a result of related attributes having different domains. The math construct was designed to alleviate this by permitting the system to transform the domain of an attribute from one representational space to another.



(a)



(b)

Figure 3.2. The background knowledge base. a) A semantic network representing interrelationships between attributes; b) Dataflow graph of $salary = hrs-worked \times wages-per-hr$.

Clustering seed

The user is able to specify the type of view desired by defining a clustering seed. This clustering seed guides the conceptual clustering process by specifying the relevant attributes of the domain of discourse and the inductive (generalization) preference criteria. The clustering seed has three clauses as discussed below. Even though its syntax resembles that of an SQL query, this language operates at a higher level of abstraction over database instances rather than over relations. The BNF of the clustering seed is:

```

<clustering seed> ::= SELECT <attr-expr-list>
                     FROM   <database-list>
                     BY     <generalization-type-list>

<attr-expr-list> ::= <attr-expr> [(log-op)<attr-expr>]+
<attr-expr> ::= <attr-name> [(rel-op) <constant>]
<log-op> ::= AND | OR | NOT
<rel-op> ::= '<' | '>' | '≤' | '≥' | '=' | '≠'
<database-list> ::= <database-name> [ ',' <database-name> ]+
<generalization-type-list> ::= <generalization-type> [ ',' <generalization-type> ]+
<generalization-type> ::= UNION '(' <attr-name> ')' |
                          CLOSE-INT '(' <attr-name> ')' |
                          AVG '(' <attr-name> ')' |
                          MAX '(' <attr-name> ')' |
                          MIN '(' <attr-name> ')' |
                          EXT-LEN '(' <attr-name> ')' |
                          NIL

<attr-name> ::= <string>
<database-name> ::= <string>

```

The **SELECT** clause is an expression of the relevant attributes of the user's domain of discourse. Several operators can be used in this expression to restrict the domain

of attributes. These operators are $\{>, <, \leq, \geq, \neq, =\}$. Boolean operators AND, OR and NOT are used to construct more complex expressions. The FROM clause is a list of databases from which the view will be generated. Only the instances of these databases will be considered. The BY clause specifies the type of generalizations to be applied to each attribute for the generation of class descriptions and attribute restrictions. The clustering algorithm abides by these types of generalizations. The types of generalization applied to an attribute depend on its domain. If the domain of the attribute is of type integer or real we can obtain the union of all the values (UNION), the range of all values by 'closing the interval' (CLOSE-INT), or perform statistical analysis of the values such as the mean (AVG), maximum (MAX) and minimum (MIN). By not specifying a type of generalization, the attribute restriction is generalized to the whole domain. A more complete explanation of these generalizations and how they affect the generation of class descriptions is given in section 3.3.3. Consider the following clustering seed:

```

SELECT teaches OR salary
FROM   Database-1, Database-2
BY     UNION(teaches), CLOSE-INT(salary)

```

This seed specifies that the attributes 'teaches' or 'salary' will be relevant for our clustering and that only instances from Database-1 and Database-2 will be considered. Furthermore, it specified to generalize over the domain of 'teaches' by obtaining the union of all its values and to apply the function of 'close-interval' to obtain a range of all the values of 'salary'.

View generator

The view generator module will generate a view of the data in the chosen data model (CANDIDE in the proposed implementation) according to the clustering seed,

background knowledge and the database instances. A detailed explanation of this module is presented in section 3.3. In the next section we describe the CANDIDE data model which is currently employed to illustrate the conceptual clustering concepts presented in this paper.

3.2.2 The CANDIDE Data Model

A CANDIDE database, discussed in greater detail in [8], consists of a set of objects $\{C, I, A, D\}$ where C are classes, I instances, A attributes and D disjoint decompositions. Each class consists of the following six-tuple

$$C = \{\text{CLASS NAME, PD-FLAG, SUPERCLASSES, SUBCLASSES, INSTANCES, ATTRIBUTE RESTRICTIONS}\}$$

CLASS NAME is a unique object identification (OID). Each class description contains pointers to its most specific superclasses, SUPERCLASSES, to the most general subclasses, SUBCLASSES, and to instances that have this class as their most specific parent, INSTANCES. The ATTRIBUTE RESTRICTIONS specify the conditions that determine class membership. If the attribute restrictions are strong enough to specify necessary and sufficient conditions for class membership, the PD-FLAG will be set to DEFINED. Otherwise, the PD-FLAG will be set to PRIMITIVE because these attribute restrictions only specify necessary conditions for class membership. Each attribute restriction is specified by

$$\text{ATTRIBUTE RESTRICTION} = \{\text{ATTRIBUTE_NAME, RESTRICTION}\}$$

Each attribute restriction, which is a form of structural cardinality, applies to a particular attribute identified by ATTRIBUTE_NAME. Furthermore, there may be several RESTRICTIONS on the values for that particular attribute. Each restriction has one of the following forms:

$$\text{SOME } n \langle v \rangle$$

EXACT $n \langle v \rangle$

ALL $\langle v \rangle$

The SOME restriction determines a minimum number of values, n , in domain $\langle v \rangle$. EXACT indicates that an attribute must have exactly n values in $\langle v \rangle$. ALL says that all the values of the attribute must be in $\langle v \rangle$.

The domains $\langle v \rangle$ can be specified using the following types and type constructors:

Simple atomic: INTEGER | REAL | STRING

Sets: RANGE | SET

Complex objects: CLASS | INSTANCE | COMPOSITE

CLASS specifies that the value of the attribute must be an instance of a class. That is, the domain of the attribute is a dynamic set of instances which constitute that class. INSTANCE specifies that the domain of the attribute is a particular instance. The COMPOSITE domain is the aggregation of other (possibly) complex domains, in which each component domain is labeled by an attribute name along with its constraints. Each instance I consists of

$$I = \{\text{INSTANCE_NAME}, \text{PARENT_CLASS_NAMES}, \text{ATTRIBUTES}\}$$

INSTANCE_NAME is a unique object identification (OID). PARENT_CLASS_NAMES is a list of the names of the most specific classes of which I is a member. ATTRIBUTES is a list of ATTRIBUTES associated with I . Each ATTRIBUTE is specified by

$$\text{ATTRIBUTE} = \{\text{ATTRIBUTE_NAME}, \text{VALUES}\}$$

The values are constructed from the same type and type constructors used in the attribute restrictions for class membership. However, the types represent the actual attribute values rather than the domain to which the value belongs. An example of a class definition of Student is shown below.

Student

DEFINED

SUPERCLASSES: Person

INSTANCES: Jim, Fred

ATTRIBUTE RESTRICTIONS

Major: SOME 1 CLASS Department

Courses: ALL CLASS Course

Age: EXACT 1 RANGE(18, 22)

Advisor: SOME 1 CLASS Professor

Disjoint compositions are lists of classes that are mutually disjoint. That is, they do not share any instances in common. Along with these constructs there exists a class hierarchy and an attribute hierarchy, shown in Fig. 3.3.

The class hierarchy shows superclass-subclass relationships among classes. The root of the class hierarchy is the universal class called **THING**. The insertion of a new class into an existing database, called classification, is performed automatically through subsumption. Class C_1 subsumes class C_2 if every instance of C_2 is a member of C_1 . In other words, C_1 subsumes C_2 if C_2 satisfies the necessary and sufficient conditions specified in the class description of C_1 . Thus, the subsumption relationship can be determined for any two classes, C_1 and C_2 by using the **SUBSUME** function:

$$\text{SUBSUME}(C_1, C_2) = \{TRUE, FALSE\}$$

which is **TRUE** if C_1 subsumes C_2 and **FALSE** otherwise. Essentially, by using the **SUBSUME** function, the classifier finds the most specific classes subsuming the new class and the most general classes subsumed by the new class. An example of a class hierarchy is shown in Fig. 3.3a.

The attribute hierarchy captures the attribute domain set/subset relationships. The root of this hierarchy is the universal attribute called THING. Each attribute can have at most one parent attribute along with an associated domain and synonym(s). The domain of an attribute must be a subset of the domain of its parent attribute. An example of an attribute hierarchy is shown in Fig. 3.3b.

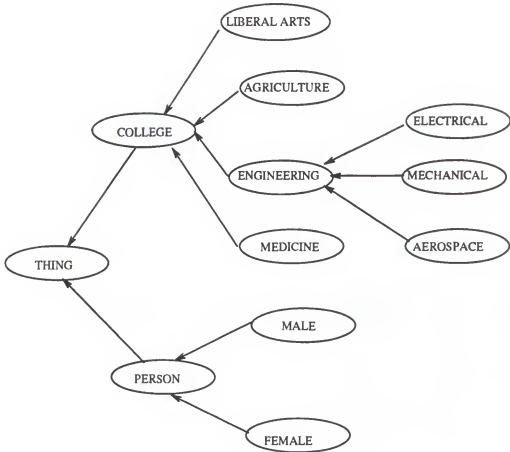
3.3 Application of the Conceptual Clustering System to Databases

In this section the algorithm employed by the view generator is considered in greater detail. For clarity, in conjunction with the algorithm, a detailed example will be explained. Consider Fig. 3.4 and Fig. 3.5 representing the instances of Database-1 and Database-2, respectively, that are connected to the CCS. The instances are in attribute-value pair vectors of variable sizes with the first string representing a unique object identification (OID). We will assume that the following facts are asserted in the background knowledge base of the system:

```
subset(teaches, noid)
subset(course-load, noid)
subset(advises, noid)
synonym(dept-name, dept)
synonym(course-taught, teaches)
logical(advises, teaches)
math(salary = hrs-worked × wages-per-hr)
```

Suppose that the user specified the following clustering seed:

```
SELECT teaches OR salary > 10000
FROM   Database-1, Database-2
BY     UNION(teaches), CLOSE-INT(salary)
```

(a)



(b)

Figure 3.3. A CANDIDE class taxonomy and attribute hierarchy. a) Class taxonomy; b) Attribute hierarchy.

By this clustering seed, the user is requesting all instances, of Database-1 and Database-2, that correspond to persons who teach or earn a salary over 10000. For those instances that qualify, the conceptual clustering algorithm will define classes accordingly. The three major steps of the algorithm are discussed in the following subsections.

3.3.1 Determination of the Relevant Instance Set and Relevant Attribute Set

From the SELECT clause of the clustering seed we are able to determine the attributes selected by the user. In the above example these attributes are ‘teaches’ and ‘salary’. Furthermore, the user specified that only the instances of Database-1 and Database-2 should be considered. An *analogy function* is employed to determine the relevant instance (RI) set and the relevant attribute set. This function considers every instance in the specified databases and adds that instance to the RI set if:-

1. any of the attributes of the instance directly match the user specified attributes or its synonyms, provided the attribute value satisfies the expression in the SELECT clause. This is considered as a direct match.
2. any of the instance attributes can be derived from one of the user specified attributes by:
 - applying different transformations (logical/arithmetic) modeled in the background knowledge base by the logical and math constructs, respectively,
 - inferring a set-subset relationship between them through the subset construct, and its value satisfies the expression in the SELECT clause.

This is considered an indirect match.

If an instance is determined to be relevant by one of the user specified attributes, that user specified attribute will be added to the relevant attribute (RA) set. At the

(John Tigert; ssn: 111; sex: 'M'; dept: "EE"; age: 20; GPA: 3.2; deg-sought: 'BS'; advisor: 'Jim Rogers'; class: 'Frsh.'; minor: 'CS'; course-load: { 'EE-101', 'CS-101', 'PSY-151', 'LIT-240' })	(Ann Jones; ssn: 222; sex: 'F'; dept: 'CS'; age: 18; GPA: 2.8; deg-sought: 'BS'; advisor: 'Jack Turner'; class: 'Soph.'; minor: 'PSY'; course-load: { 'CS-101', 'PSY-151', 'HIS-145', 'LIT-240' })	(Amy Allen; ssn: 333; sex: 'F'; age: 20; dept: 'CS'; GPA: 3.1; deg-sought: 'BS'; advisor: 'Jack Turner'; class: 'Jun.'; minor: 'EE'; course-load: { 'EE-101', 'CS-101', 'CS-125', 'PSY-151' })
(Jack Andrews; ssn: 444; sex: 'M'; age: 25; dept: 'EE'; GPA: 3.8; class: 'Grad.'; deg-sought: 'MS'; res-topic: 'Control Systems'; chairman: 'Jim Rogers'; course-taught: 'EE-101'; hrs-worked: 30; wages-per-hr: 7.00; course-load: { 'EE-541', 'EE-572', 'CS-589' })	(Jerry Smith; ssn: 555; sex: 'M'; age: 27; dept: 'EE'; class: 'Grad.'; GPA: 3.76; deg-sought: 'Ph.D'; res-topic: 'Adaptive Filters'; hrs-worked: 30; chairman: 'Jim Rogers'; course-load: { 'EE-541', 'EE-572', 'CS-627' })	(Daniel Murphy; ssn: 666; sex: 'M'; age: 30; dept: 'CS'; GPA: 3.9; class: 'Grad.'; deg-sought: 'Ph.D'; res-topic: 'Distr. DBMS'; chairman: 'Jack Turner'; course-taught: 'CS-101'; hrs-worked: 20; wages-per-hr: 10.00; course-load: { 'CS-589', 'CS-627', 'EE-572' })
(Jim Rogers; ssn: 777; sex: 'M'; age: 45; dept: 'EE'; salary: 45,000; pos: 'Assist. Prof'; societies: 'IEEE', 'IEE'; teaches: 'EE-541', 'EE-572'; advises: { 'Jerry Smith', 'Jack Andrews', 'John Tigert' })	(Jack Turner; ssn: 888; sex: 'M'; age: 52; dept: 'CS'; salary: 55,000; pos: 'Prof.'; societies: 'ACM'; teaches: 'CS-627', 'CS-589'; advises: { 'Daniel Murphy', 'Ann Jones', 'Amy Allen' })	(Ann Doe; ssn: 999; sex: 'F'; age: 35; dept: 'EE'; salary: 25,000; pos: 'Secretary'; union: 'United Secretaries of America')

Figure 3.4. Instances of Database-1

(EE-101; course-name: 'Intro. to Eng.'; dept-name: 'EE'; credit: 3)	(CS-101; course-name: 'Intro. to C.S.'; dept-name: "CS"; credit:3)	(PSY-151; course-name: 'Child Psy'; dept-name: 'PSY'; credit: 3)
(HIS-145; course-name: 'Egyptian His.'; dept-name: 'HIS'; credit: 3)	(LIT-240; course-name: 'Modern Poetry'; dept-name: 'ENL'; credit: 3)	(CS-125; course-name: 'Databases'; dept-name: 'CS'; credit: 3)
(EE-541; course-name: 'Signal Proc.'; dept-name: 'EE'; credit: 3)	(EE-572; course-name: 'Pattern Rec.'; dept-name: 'EE'; credit: 3)	(CS-627; course-name: 'Algorithms'; dept-name: 'CS'; credit: 3)
	(CS-589; course-name: 'Operating Systems'; dept-name: 'CS'; credit: 3)	

Figure 3.5. Instances of Database-2

end of this step we would have determined all the relevant instances and relevant attributes.

Formally, the analogy function is a mapping from the set of database instances specified in the FROM clause, I_{DB} , and the set of attributes specified in the SELECT clause, A_{SEL} , to the RI set and the RA set. Consider,

$$\Psi = \{I_{DB}, A_{SEL}\}$$

and

$$\Omega = \{RI, RA\}.$$

The analogy function, Φ , maps Ψ to Ω as follows:-

$$\Phi : \Psi \longrightarrow \Omega$$

such that

$$\begin{aligned}
 & \text{IF } x \in I_{DB} \wedge \\
 & \quad attr_j \in A_{SEL} \wedge \\
 & \quad \exists attr_i \{ \\
 & \quad \quad attribute-of(attr_i, x) \wedge \\
 & \quad \quad satisfies-expr(attr_i) \wedge \\
 & \quad \quad \{ attr_i \equiv attr_j \vee \\
 & \quad \quad \quad subset(attr_i, attr_j) \vee \\
 & \quad \quad \quad synonym(attr_i, attr_j) \vee \\
 & \quad \quad \quad logical(attr_i, attr_j) \vee \\
 & \quad \quad \quad satisfies-math-expr(x, attr_j) \\
 & \quad \quad \} \\
 & \quad \} \\
 & \text{THEN } x \in RI \wedge attr_j \in RA
 \end{aligned}$$

where *attribute-of(attr, x)* is true if *attr* is an attribute of instance *x*, *satisfies-expr(attr)* is true if the value of the attribute satisfies the expression in the SELECT clause and *satisfies-math-expr(x, attr)* is true if there exists a fact in the knowledge base that models a mathematical relationship between *attr* and the attributes of instance *x*.

From our example, ‘Jack Andrews’, ‘Daniel Murphy’, ‘Jim Rogers’, ‘Jack Turner’ and ‘Ann Doe’ are considered relevant, as shown in Fig. 6. Attributes that caused the relevance to be established are marked by an asterisk in the figure. These instances were considered relevant for more than one reason. All these instances are considered relevant because they have attributes that directly match or can derive the relevant attribute ‘salary’ and the attribute value exceeded 10000. ‘Jim Rogers’, ‘Jack Turner’ and ‘Ann Doe’ have the attribute ‘salary’, which is a direct match. ‘Jack Andrews’ and ‘Daniel Murphy’ have the attributes ‘wages-per-hr’ and ‘hrs-worked’ that can be transformed into ‘salary’ by the fact *math(salary = wages-per-hr × hrs-worked)*. Notice that ‘Jerry Smith’ was not considered relevant because ‘salary’ could not be derived from just ‘hrs-worked’. Also, ‘Jack Andrews’ and ‘Daniel Murphy’ are relevant because they possess the attribute ‘teaches’ and because the attribute

‘course-taught’ is a synonym of ‘teaches’ in *synonym(course-taught, teaches)*. ‘Jim Rogers’ and ‘Jack Turner’ are considered relevant because ‘advises’ implies ‘teaches’ as modeled by *logical(advises, teaches)*. Notice that the RI set is not affected by the presence of the attribute ‘teaches’ in the clustering seed. This is reasonable as anybody who teaches will usually get a salary. Since both the attributes specified by the user in the SELECT clause were used in determining the relevant instances, they were added to the relevant attribute (RA) set. This eliminates attributes that did not affect the determination of the RI set. Thus, the RA set consists of {*teaches, salary*}.

3.3.2 Generation of the Initial Class Taxonomy and Derived Attribute Set

At this stage we have determined the relevant instance (RI) set and the relevant attribute (RA) set. Now the initial class taxonomy can be constructed. This process consists of the following steps:

1. Determine the largest common subexpression of attributes among the instances of the RI set. This is a prototypical representation of the instances of the RI set. A class with these attributes is created. Add the attributes of the subexpression, if any, to the *derived attribute (DA) set*. Note, this subexpression might be null because as the number of instances grow they will have very little in common (this is termed as the ‘family resemblance effect’). All of the instances of the RI set are members of this class. This class is a subclass of THING.
2. For each class generated, C, determine the largest, non-overlapping subexpressions of attributes, not members of the DA set, among instances of that class. In choosing an attribute subexpression we try to maximize
 - the number of instances ‘covered’ by each subexpression and
 - the length of the subexpression.

These subexpressions should cover all the instances of the class unless all the attributes of an instance are already members of the DA set. A candidate

(Daniel Murphy;	(Jim Rogers;	(Jack Turner;
ssn: 666;	ssn: 777;	ssn: 888;
sex: 'M';	sex: 'M';	sex: 'M';
age 30;	age: 45;	age: 52;
dept: 'CS';	dept: 'EE';	dept: 'CS';
GPA:3.9;	* salary: 45,000;	* salary: 55,000;
class: 'Grad.';	pos: 'Assist. Prof';	pos: 'Prof.';
deg-sought: 'Ph.D';	societies: 'IEEE', 'IEE';	societies: 'ACM';
res-topic: 'Distr. DBMS';	* teaches: 'EE-541', 'EE-	* teaches: 'CS-627', 'CS-
chairman:'Jack Turner';	572';	589';
* course-taught: 'CS-101';	* advises: {'Jerry Smith',	* advises: {'Daniel Murphy',
* hrs-worked: 20 ;	'Jack Andrews',	'Ann Jones',
* wages-per-hr: 10.00;	'John Tigert'})	'Amy Allen'})
course-load: {'CS-589',		
'CS-627', 'EE-572'})		
	(Jack Andrews;	(Ann Doe;
	ssn: 444;	ssn: 999;
	sex: 'M';	sex: 'F';
	age: 25;	age: 35;
	dept: 'EE';	dept: 'EE';
	GPA: 3.8;	* salary: 25,000;
	class: 'Grad.';	pos: 'Secretary';
	deg-sought: 'MS';	union: 'United Secretaries
	res-topic: 'Control Systems';	of America')
	chairman: 'Jim Rogers';	
	* course-taught: 'EE-101';	
	* hrs-worked: 30;	
	* wages-per-hr: 7.00;	
	course-load: { 'EE-541',	
	'EE-572', 'CS-589'})	

Figure 3.6. Relevant instances

class is created for each subexpression. Instances with these subexpressions are members of the corresponding candidate classes. A candidate class is added to the class taxonomy if an identical class, with the same instances, does not already exist. The resultant classes generated are subclasses of the original class, *C*, and instances could participate in more than one class. All the attributes of the subexpressions are added to the DA set.

3. Repeat step 2 until all the attributes in the RA set are accounted for in the DA set. At the end of this process, the DA set and the initial class taxonomy will be completely specified.

In our example, the largest common subexpression of attributes among the RI set is {'ssn', 'dept', 'sex', 'salary', 'age'}. Notice that, even though the attribute 'salary' is not actually common to all the instances, it can be derived from other attributes of the instance description by using the fact $math(salary = wages-per-hr \times hrs-worked)$. A class is created with these attributes. This class is named EMPLOYEE after examining the instances that belong to it (this will be in consultation with the user). Naming is of no consequence as it does not affect class descriptions and inferencing [12]. The largest, nonoverlapping attribute subexpressions of the instances of EMPLOYEE are {'teaches'} and {'pos'}. Consequently, two classes corresponding to these attribute subexpressions are created. These classes are INSTRUCTORS and FULL-TIME-EMP, respectively. 'Jim Rogers', 'Jack Turner' and 'Ann Doe' are members of FULL-TIME-EMP. 'Jim Rogers', 'Jack Turner', 'Jack Andrews' and 'Daniel Murphy' are members of INSTRUCTORS. At this stage we realize that all the attributes of the RA set have been accounted for in the DA set. The DA set is {'ssn', 'dept', 'sex', 'salary', 'age', 'pos', 'teaches'}. Notice how the system was able to use default reasoning to predict missing information. The user just specified the attributes 'salary' and 'teaches', but the system was able to determine that the attributes 'ssn', 'dept', 'sex', 'age' and 'pos' are also pertinent. Notice that other

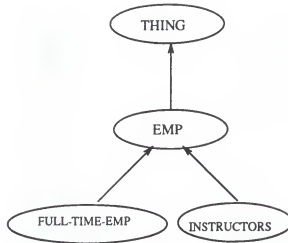


Figure 3.7. The initial taxonomy

attributes of the instances were not added to the DA set, such as 'res-topic', 'GPA', and 'class' of 'Jack Andrew' and 'Daniel Murphy'. This is because these attributes reflect specific aspects of the instances not captured in the level of abstraction described by the user. The user in the clustering seed essentially specified a view of all who either get salaries or teach. The specifics of these classes were not considered relevant by the user. The initial class taxonomy is shown in Fig. 3.7.

3.3.3 Determination of Attribute Restrictions and Final Class Taxonomy

At this stage the initial taxonomy has been built and we can perform induction over the instances of the classes according to specific generalization rules specified by the user in the clustering seed. The system incorporates several generalization rules for each type of attribute domain. The attribute domains supported by the CANDIDE semantic data model are integer, real, string, object identifier (OID), multi-valued and composite. Several generalization rules can be applied to integer and real domains. We can obtain the union of values (UNION), the range of all values by 'closing the interval' (CLOSE-INT), or perform statistical analysis of the attribute values such as the mean (AVG), the maximum (MAX) and minimum (MIN). There

are two types of generalizations for string domains. These are obtaining the union of all values and extending the length of the string (EXT-LEN).

If the domain of the attributes is a set of instances (OID), all the instances referenced by the attribute are combined into a set. This set will constitute a class. If an identical class (one with the same instances) already exists, that class will be the domain of the attribute, otherwise a new class of these instances will be placed in the taxonomy by applying the subsumption function. That is, we will determine the set-subset relationships between this class and other classes in the taxonomy. Only the attributes of the instances of this class that are members of the DA set will be considered because it was these attributes that were determined relevant. Composite attributes are attributes that are an aggregation of other (possibly aggregate) attributes. Generalization will proceed over the atomic components of these attributes as mentioned above.

From multi-valued attributes, we are able to induce several structural cardinality constraints. If all the attributes have the same number of values, we will be able to generate the following constraint

$$\text{EXACT } n \langle v \rangle$$

where $\langle v \rangle$ is the domain of values and n is the number of those values. If the attribute has different numbers of values we can determine the minimum number and specify the following constraint

$$\text{SOME } n \langle v \rangle$$

where n is that number. If all the attribute values are from the same domain we can specify the following constraint

$$\text{ALL } \langle v \rangle$$

Another structural cardinality which induces the maximum number of values an attribute can have (MAX) can also be generated. But this cardinality is not supported by CANDIDE to ensure tractability of the subsumption function.

In the example we are considering, the DA set is {'ssn', 'dept', 'sex', 'salary', 'age', 'pos', 'teaches'}. The only attribute of the DA set with a domain of instances is the 'teaches' attribute. This is determined by $\text{subset}(\text{teaches}, n_{oid})$. All the instances referenced by this attribute are grouped into a set. This set will constitute the class COURSE. This set of instances is shown in Fig. 3.8. After applying the subsumption function to this class, to determine the place of this class in the class taxonomy, it was determined that the only class that subsumes COURSE is THING. Here the use of the subsumption function is to determine set-subset relationships between the instances of the classes. The final taxonomy is shown in Fig. 3.9. The only attribute of the class COURSE that will be considered is 'dept', because it is the only member of the DA set. Since the user only specified the type of generalization for 'teaches' and 'salary' in the clustering seed, he will be prompted to specify the types of generalizations for the other attributes of the DA set. If the user specifies the UNION option for 'sex', 'dept' and 'pos', the CLOSE-INT option for 'age', and none for 'ssn', the class descriptions shown in Fig. 3.10 will result. This is the view generated by the system. Notice how the structural cardinality constraints of the same attribute differ from class to class. The attribute 'salary' of EMP ranges from 10800 to 55000, while for FULL-TIME-EMP it ranges from 25000 to 55000. The attribute 'age' for EMP ranges from 25 to 52, while for FULL-TIME-EMP it ranges from 35 to 55.

The degree of restriction of the generated integrity constraints is controlled by the type of generalization specified. If the user does not specify a type of generalization, all values of the domain of the attribute will be permitted. That is, the valueset constraint will be generalized to the entire domain of the attribute. By specifying the CLOSE-INT (closing the interval) option, any value within the maximum and minimum values will be permitted. The UNION option will restrict values to those already in the database. Notice how some generalizations do not make much sense such as closing the interval (CLOSE-INT) for 'dept#', assuming the domain of

(EE-101; course-name: 'Intro. to Eng.'; dept-name: 'EE'; credit: 3)	(CS-101; course-name: 'Intro. to CS'; dept-name: 'CS'; credit: 3)
(EE-541; course-name: 'Signal Proc.'; dept-name: 'EE'; credit: 3)	(EE-572; course-name: 'Pattern Rec.'; dept-name: 'EE'; credit: 3)
(CS-627; course-name: 'Algorithms'; dept-name: 'CS'; credit: 3)	

Figure 3.8. Instances of COURSE referenced by 'teaches'

'dept#' was integer. The type of generalization used is left to the user's discretion. By selectively applying these generalizations, different views can be generated for different user groups. Views for naive users, such as data entry operators, will typically be more restrictive to detect data entry errors. On the other hand, views for more sophisticated user groups, such as management and the database administrators, will probably be less restrictive to permit more flexible access and update of the database.

3.4 Implementation Issues

The algorithm we have presented is computationally expensive primarily due to the inferencing required to determine attribute relevancy and the potential search problem required to retrieve instances from the database. To tackle these problems we

- compile the background knowledge into an efficient structure for determining attribute relevancy,
- physically organize the database into *conceptual categories* [32] to permit efficient access of instances.

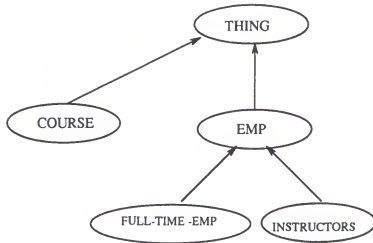


Figure 3.9. The final taxonomy

The background knowledge base is compiled by a special purpose theorem prover as a once-only compiler into a compiled relevancy network (CRN). The basic idea is as follows. For each attribute known to the system, determine all other attributes that can be deduced from the former. This means at execution time there is no need for inferencing in determining attribute relevancy, hence the problems involving searching can be avoided. The CRN is an indexing scheme in which each attribute is associated with a set of relevant attributes, the compiled relevant attribute (CRA) set. Thus, once an attribute is specified in the SELECT clause, all the attributes that can be derived from it can be immediately determined. Fig. 3.11 shows the CRN for the set of facts in the background knowledge base of the previous example. Notice for 'salary' how 'hrs-worked' and 'wages-per-hr' are *jointly* relevant. That is to say, both these attributes are required to determine 'salary' as defined in the fact $math(salary = hrs-worked \times wages-per-hr)$. This is modeled in the CRN by 'hrs-worked & wages-per-hr'.

The database is physically organized into conceptual categories by an inverted indexing procedure in which there is a category corresponding to each attribute known to the system. That is, there exists a category for each attribute, such that all the instances of that category possess that attribute. Moreover, categories are also

EMP

DEFINED

SUPERCLASSES: THING

INSTANCES: 'Jack Andrews'
 'Daniel Murphy'
 'Jim Rogers'
 'Jack Turner'
 'Ann Doe'

ATTRIBUTE RESTRICTIONS:

ssn: EXACT 1 INTEGER
 sex: EXACT 1 SET('M', 'F')
 salary: EXACT 1
 RANGE(10800,55000)
 dept: EXACT 1 SET('CS', 'EE')
 age: EXACT 1 SET(25, 52)

FULL-TIME-EMP

DEFINED

SUPERCLASSES: EMP

INSTANCES: 'Jim Rogers'
 'Jack Turner'
 'Ann Doe'

ATTRIBUTE RESTRICTIONS :

ssn: EXACT 1 INTEGER
 sex: EXACT 1 SET('M', 'F')
 age: EXACT 1 RANGE(35, 52)
 salary: EXACT 1
 RANGE(25000, 55000)
 dept: EXACT 1 SET('CS', 'EE')
 pos: EXACT 1 SET ('Prof.',
 'Asst. Prof.',
 'Secretary')

INSTRUCTORS

DEFINED

SUPERCLASSES: EMP

INSTANCES: 'Jack Andrews'
 'Daniel Murphy'
 'Jim Rogers'
 'Jack Turner'

ATTRIBUTE RESTRICTIONS:

ssn: EXACT 1 INTEGER
 sex: EXACT 1 SET('M', 'F')
 age: EXACT 1 RANGE(25, 52)
 salary: EXACT 1
 RANGE(10800, 55000)
 dept: EXACT 1 SET('EE', 'CS')
 teaches: SOME 1 CLASS COURSE
 ALL CLASS COURSE

COURSE

DEFINED

SUPERCLASSES: THING

INSTANCES: 'EE-101'
 'CS-101'
 'CS-627'
 'EE-541'
 'EE-572'

ATTRIBUTE RESTRICTIONS:

dept: EXACT 1 SET('EE', 'CS')

Figure 3.10. The generated class descriptions

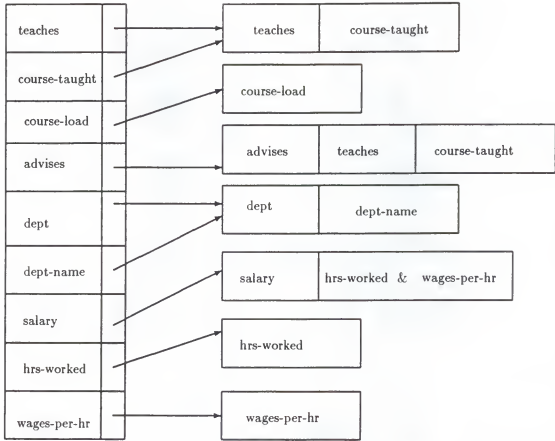


Figure 3.11. The compiled relevancy network for the previous example

LEVEL 0

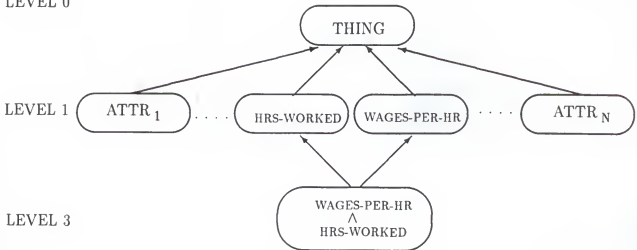


Figure 3.12. The organization of the database

created for sets of attributes that are determined to be jointly relevant. Each of these categories index all their members for efficient retrieval. Thus, once an attribute is determined to be relevant to the clustering seed, all the instances that possess that attribute are directly retrieved from the corresponding category. In such an organization, the category system will comprise at most three levels. Fig. 3.12 illustrates this by using the previous example. Notice how the category 'hrs-worked \wedge wages-per-hr' is a subclass of both 'hrs-worked' and 'wages-per-hr'.

CHAPTER 4

SCHEMA EVOLUTION AND EXCEPTION HANDLING TECHNIQUES

Traditionally, the information stored in a database system has been partitioned into schema and facts. The schema is a generic and typically time-invariant representation of the concepts modeled by the database system. Facts, on the other hand, are volatile and occur in unproportionally large numbers. Unfortunately, the schema of a database system may not reflect entirely accurately the corresponding aspects of the real world because schema design is still considered an art rather than a science and is often modeled by by someone who is unfamiliar with the application domain. The error-prone process of interviewing end-users to determine relevant concepts and the irregularity, evolution, variability and complexity of the domain of discourse compounds this problem [10]. In particular, the process of generating *necessary properties* for objects belonging to a certain class or integrity constraints that apply to a certain class are very difficult if not impossible when defining 'natural kinds'.

All the factors mentioned above may lead to the possible loss of information for several reasons. Due to the *rigidity* of conventional database systems, instances that do not conform to the schema are either tailored to it or excluded altogether. This is because exceptional values are not tolerated in a database. Once the integrity constraints of a database system are violated, the entire system is rendered inconsistent. In addition, predefined application programs may produce nonsensical results [10].

Typically, schema designers have opted to design the schema in *very general* terms in order to alleviate these problems. But, this constitutes an abdication of their responsibility to accurately design the domain of discourse. In this chapter we propose that a database management system be endowed with the capability to refine its schema on the basis of the data actually stored so far. A conceptual clustering algorithm based on a theory of categories aids in building classes by grouping related

instances and developing class descriptions. Class cohesion results from a tradeoff between class descriptions and instance similarity. This is achieved by combining techniques from machine learning, specifically explanation-based learning and case-based reasoning. A new function, INTERSECT, is introduced to compare the similarity of two instances. INTERSECT is used in defining an exception condition. Exception handling, in turn, results in schema modification.

The rest of the chapter is organized as follows. In the next section we will review the prominent database systems that allow for schema evolution. Section 4.2 will discuss in detail our notion of exception as compared to others. In section 4.3 a detailed discussion of the INTERSECT function is presented. Section 4.4 presents sheds some light on performance issues of the INTERSECT function.

4.1 Comparison to Related Work

Borgida et al. [10] investigated the possibility of having automatic aids that identify cases in which a number of exceptions mounts to a point where a change in the schema is indicated and then suggest to the DBA possible improvements. In particular, two techniques were presented through which a computer system can suggest modifications and additions to the schema and semantic integrity constraints of a database by learning from exceptions. The first method refines the schema by generalizing from *example* exceptions to form descriptions of classes of similar objects. The second method refines integrity constraints through *explanation-based learning* by utilizing a detailed theoretical world-model and then generalizing to those cases where similar explanations hold.

An algorithm *Descr* was introduced that infers the most specific class description(s) of a set of instances. *Descr* takes as input a set of instance descriptions and produces a class description which has as those instances as its members. This should be the most specific such class, in the sense that any other class that would have these instances as members would subsume this class. Initially, the description

of the exceptions is initialized to the first exceptional instance. The next exceptional instance is compared to this description. Whenever part of the class description is too restrictive, that part is generalized by one of the rules below. In essence this algorithm makes a *specific-to-general breadth-first search* to induce a description for a set of objects. The algorithm consists of the following steps:

1. Introduce/Expand Range. A range can be introduced or expanded to cover a new scalar data type. For example, to describe salaries of 40,000 and 60,000 respectively, the attribute specification *wages: 40000 .. 60000* would be necessary.
2. Introduce/Generalize Class. An existing class can be introduced or generalized (using the IsA hierarchy) in a description to cover a new data value, especially an entity object. For example, *supervisor : MANAGER* can be generalized to *supervisor: EMPLOYEE*. In this generalization one uses the most specific parent of the class.

The *Descr* function has several shortcomings. Since only only exceptions are considered, classes generated by the algorithm will represent an intensional description of the similarity exhibited by only those exceptional instances. The similarity that may exist between these exceptional instances and other instances of the database is ignored. The *Descr* may also generate spurious restrictions based on accidental commonalities in data, unless a very large sample of objects is considered. The presence of such spurious attribute specifications is problematic for two reasons: it makes integrity constraints more expensive to check and it allows fewer objects to be present, thus circumventing our goal of the need to excuse exceptional cases. These two problems can be resolved by selectively eliminating all but the most 'essential' attribute specification in the description. For this purpose, a notion of *relevance* was

introduced. This is an evaluation of which attribute specifications are more relevant to the generalization at hand. Borgida introduced some heuristics to determine relevance including

- Classes, and their properties which are closer to a particular class in the hierarchy are more relevant to it than ones further away.
- Classes and properties mentioned in the integrity constraints are likely to be relevant.
- When describing objects that are exceptional to a certain property, than that property is not relevant.
- Statistical distribution or frequency of certain values can be used. For example, if certain values that are not very frequent in the data base are absent, then their absence is not significant.

Borgida et al. [10] also discussed the use of an explanation-based learning algorithm for learning from examples. This technique utilizes domain-specific knowledge to explain why a certain situation has occurred and then to extract from this explanation the salient features which generalize to other similar situations. The key to such an approach is the presence of a *theory* for generating explanations. This theory expresses the semantics of the terms used in the database. This type of knowledge is not normally available in standard database systems as data dictionaries are not expressed formally. Because such theories are very complex is not feasible to check if the database is consistent with these theories. Thus, integrity constraints of the database can be considered compiled rules of judgment which can be verified efficiently and whose role is to detect errors which are expected to occur. These integrity constraints may however be too restrictive and when new and unusual circumstances occur it is the role of the system to generalize the integrity constraints as much as necessary.

Considerable work has been done on schema evolution in object-oriented database systems [38, 44]. The thrust of this work was to allow database systems to dynamically change object definitions and to incrementally define composite objects. Specifically, the database system should be endowed with the ability to support evolving schema and to propagate the changes on the object instances. Current results indicate that while most database systems provide schema evolution facilities, they seldom support automatic propagation mechanisms [44]. Three different schema change operations were identified [44]. The first, changing class definitions, encompasses the addition or deletion of instance variables or methods to or from a class definition. Moreover, these instance variables and methods may be modified. Secondly, the class lattice may be modified by determining new superclass or subclass relationships. Finally, classes may be deleted or added to the class lattice. It should be noted that no classification mechanism is used in object-oriented systems; that is, there is no automatic means by which to determine superclass, subclass or instance-of relationships. In addition, these changes are either committed whenever the object is retrieved (screening) or instantly (conversion). GOOSE [38], on the other hand, uses a hybrid approach as it propagates changes instantly to main memory objects which are made permanent when saved.

Composite objects are basically structured aggregates of the sub-parts involving the definition of other objects. A dependency relationship exists between the components and the 'owner'. Moreover, this relationship must be provided if the semantics of the composite object is to be applied on the components. Therefore, changes to the definition of the composite objects must be propagated to their components (deleting the dependency relationship). Similarly, changes to the definition of the component must be visible to the owner to enforce this dependency relationship.

Basically, all the changes or evolution in object-oriented systems is manual, it has to be performed by the user. Most of the literature, if not all, refer to all possible evolutionary modifications that are permitted by one system or the other. None,

to the knowledge of the author, address the fundamental problem of detecting when evolution should occur, what causes evolution and how this process can be automated. This point was only alluded to [44] when classification was proposed as a means to automate this process.

4.2 Exception Handling

Exception handling has emerged as a particularly difficult problem, and one for which we have not yet developed a satisfactory solution. This is due in part to the extremely rigorous way we treat exceptions. Previous treatments of exceptions have, in our view, failed to appreciate the difficulty of the problem and defined exceptions in a relatively trivial fashion. In fact, exceptions play a central role in the dynamics and evolution of categories. The category system must modify itself to accommodate exceptions, but first it must be able to recognize exceptions. While category modification, which data modelers call schema evolution, is itself not trivial, we argue that exception recognition is much more difficult [15, 21, 24].

Intuitively, exceptions occur whenever the classification system makes an error. For example, a disease diagnosis program fails to classify a disease correctly, a computer vision system mistakenly identifies a telephone pole as a tree, a bird mistakes a broom handle for a snake, a policeman mistakes a toy gun for a real weapon, or a language speaker misinterprets the use of a particular utterance. First notice that such examples are widely available. Such errors are not necessarily good or bad. To err is human, and computers ought to fail just as gracefully. Also, the error may never be detected, it may have no consequence. The correct disease may never be known, it may not matter to the robot whether something is a telephone pole or tree, and the bird may flee from the broom and live another day just the same. So what makes these misclassifications erroneous?

More formally, an instance is an exception if it belongs in a particular category, yet fails to meet any intensional description specifying conditions for membership in

that category. Conversely, an instance is an exception if it does satisfy conditions for membership in a particular category, yet it does not belong in that category. In either case, the exception is miscategorized. The first kind of exception is an error of excluding an instance from a category to which it belongs, the second kind is an error of including an instance in a category to which it does not belong. Failure to diagnose a case of pneumonia as pneumonia is an example of the first kind of exception. Mistaking a broom handle for a snake is an example of the second kind of exception.

The problem with previous treatments of exception handling is that they presuppose that the categories to which an instance belongs is already known. This begs the question in categorization problems. Once the category is known, it is then a simple matter to test whether the intensional description is or is not satisfied. Borgida [10] uses this approach in a system which automatically learns constraints for database classes. Exceptions are given to the system, and constraints are modified to accommodate exceptions. This addresses the schema evolution problem, but does not address the problem of identifying exceptions since the classes to which the exceptions belong are told to the system. More recently, Bergadano et al [9] have developed a two-tiered concept representation which includes a Base Concept Representation (BCR) which is much like a prototype or typical properties associated with a category, and an Inferential Concept Interpretation (ICI) which provide rules and conditions for deviations from the BCR. Some exceptions can be identified by deductive reasoning from the ICI. While this uses a more sophisticated inferencing approach to identify exceptions, it could not in general identify all exceptions, specifically those which cannot be deduced from the ICI. Also, the ICI is not learned but specified by an outside trainer. In general, we have not found any approaches which consider the more difficult problem of exception handling. Making an error in an initial categorization is natural and expected, but then the error must be detected and the category structure corrected to accommodate the exception. And all this

must happen without the system being explicitly told about the error. Although learning by being told is a valid form of learning, it cannot be used as the sole means of exception handling.

In our own work we have developed a notion of similarity based on conceptual measures. Central to our similarity definition is a function which we call INTERSECT. INTERSECT takes two instances as arguments and produces a new class. The new class tells what if anything the structure of the two instances have in common by doing a graph intersection on the semantic network representations of the two concepts. Thus, the INTERSECT of London and Paris is a new class, which might be something like 'Capital Cities of European Countries'. We have attempted to develop an exception condition using INTERSECT. The exception conditions suggests that a new instance, *I*, might be an exception to an existing class, *C*. It works by computing INTERSECT between the new instance, *I*, and every instance in the database (although this sounds computationally complex, the candidate set of instances which actually need to be checked is only a small percentage of the complete database). If in the process INTERSECT generates non-trivial results between *I* and instances of *C*, yet *I* fails to satisfy the intensional description for *C*, and in addition whatever *I* has in common with other instances of *C* it does not share in common with non-members of *C* (*I* is uniquely similar to other members of *C*), then perhaps *I* is an exception to *C*. Unfortunately, 'perhaps' is the strongest that this can be stated. There is no absolute test, according to this approach, which would determine strongly that *I* is an exception, for perhaps *I* is a member of *C*, and perhaps it is not (the disjunction problem).

It appears that deductive reasoning cannot be the answer to exception handling. Exceptions cannot be identified on the basis of past and present information consisting only of empirical observations and rules for determining category membership. Given only that information, the categorization process will produce a particular

classification. If that classification is erroneous, the errors cannot be detected using the same information used to generate the classification.

So a promising direction is to look at the influence of future information on the dynamics of category structure. Simply put, a particular categorization is used to make predictions about the future, and these predictions can be tested when additional information is obtained. If the predictions turn out to be false for particular instances, then these instances are identified as exceptions and the category structure can then be modified to correct the error. This is nothing less than the scientific method. But unfortunately, again this is not a trivial solution to the problem. For how is the interpretation of the new information to be trusted? Where in the chain of reasoning did the real error occur [16]. Furthermore, it leads to the need to maintain multiple hypotheses, and that can produce a combinatorial explosion even in small domains.

In this section we have attempted to show that exception handling is not a trivial problem. Especially, a rigorous notion of what an exception is needs to be accepted. A good solution to exception handling is currently lacking. This problem stands in the way of a good conceptual clustering algorithm. In spite of this, several promising conceptual clustering techniques have been developed. These will now be described.

4.3 Incremental Clustering Algorithm

The purpose of the INTERSECT function is to determine whether two instances have anything in common by generating a class description that identifies those commonalities. That is,

$$\text{INTERSECT}(I_1, I_2) = C$$

C is created from the components of I_1 and I_2 which share some commonality. C is a minimal class description which is satisfied by both I_1 and I_2 . Thus, the INTERSECT function is used as a basis for similarity. The main components of the algorithm are shown in Fig. 4.1.

1. Introduce a new class

- (a) Use SUBSUME and CLASSIFICATION to determine the relationship between the new class and the existing class descriptions.
- (b) Use COMPLIES and Realization to determine which instances satisfy class descriptions.

2. Introduce a New Instance

- (a) Use COMPLIES and Realization to place the new instance into classes for which the instance satisfies their class descriptions.
- (b) Use INTERSECT to identify other related instances.
- (c) Use the Exception Condition to see if the new instance may be an exception to an existing class description.
- (d) Based on a decision to place an exception into a class, use EVOLVE to modify the class schema.

Figure 4.1. Main components of the conceptual clustering algorithm

To define INTERSECT, it is convenient to view each instance as a connected, possibly cyclic, directed graph having a root node. INTERSECT can then be defined as graph-matching function. Initially, an Immediate Description Graph is generated. This IDG of an instance is just those components of the graph that are identified in the instance object. From the IDG the Extended Description Graph is generated. The EDG is obtained by recursively expanding the database objects identified within the IDG of the instance. In essence, the EDG identifies all the objects in the database that are connected to the instance. Given the EDG graphs of two instances, the $\text{INTERSECT}(I_1, I_2)$ is obtained by creating a new EDG that is a graphical intersection of the two instance EDGs. This EDG is then converted into a class description. The four steps of the INTERSECT function are detailed in the following subsections. Moreover, the instances shown in Fig. 4.2 will be utilized to illustrate the algorithm.

Jim	Fred
PARENTS: Student, Man	PARENTS: Student, Man
ATTRIBUTES	ATTRIBUTES
Major: Computer Engineering	Major: Electrical Engineering
Courses: Data Structures, Pascal, Calculus	Courses: Probability, Physics
Age:19	Age:19
Advisor: John	Advisor: Joe
Residence: COMPOSITE	Residence: COMPOSITE
Type: Dormitory	Type: Apartment
Location: Collegeville	Location: Collegeville

Figure 4.2. Example database instances.

4.3.1 Generation of the Immediate Description Graph (IDG)

The IDG of an instance I is a directed graph $IDG = \{V, E, R\}$, with nodes V , arcs E , and a root R . V is partitioned into $\{R, V_p, V_s, V_v\}$ where V_p is a set of nodes corresponding to the PARENTS of I , V_s is a set of value set nodes, and V_v is a set of value nodes. E can be partitioned into $\{E_p, E_a, E_v\}$ where E_p is a set of PARENT arcs, E_a a set of ATTRIBUTE arcs, and E_v a set of VALUE arcs. The IDG is constructed as follows:

1. Construct a root node R which contains the INSTANCE NAME of I .
2. PARENT arcs are created such that one PARENT arc in E_p going out from R is created for each of I 's PARENTS. Each of these PARENT arcs are connected to node in V_p containing the CLASS NAME for that parent.
3. ATTRIBUTE arcs are created such that each attribute of I is represented by one arc from E_a labeled by the ATTRIBUTE NAME. Each attribute arc is connected to a value set node, V_s . The value nodes have outgoing arcs of E_v , labeled VALUE, for each attribute value. Each VALUE arc is connected to a value node in V_v that are in turn roots of yet another directed graph.
4. Graphs are then created for each attribute value. A value of type STRING, INTEGER or Real is stored in a single terminal node in V_v . Even though the

type RANGE contains an internal structure (minimum and maximum value) it is not expanded further and is stored in a single terminal node in V_v . A single node in V_v containing the INSTANCE NAME is created for each value of type INSTANCE. A value of type COMPOSITE is connected to an IDG corresponding to that value. A value of type CLASS is treated by replacing the class with all of its instances. That is, the single value arc for the class is removed and new VALUE arcs are created for each instance of the class.

An example of an IDG is shown in Fig. 4.3; the IDG constructs are in bold.

4.3.2 Generation of the Extended Description Graph (EDG)

The EDG of an instance I is a directed graph $EDG = \{V, E, R\}$. V and E are partitioned as in the IDG. The EDG is constructed as follows:

1. The root of the EDG is that of the IDG. V and E contain the nodes and arcs of the IDG.
2. Ancestor classes are added by considering the class generalization hierarchy as a directed graph $H = C, S, T$ where C is the set of nodes containing one node for each class in the database (the node containing the class name), S is the set of directed arcs containing one arc from each class node in C to each of the immediate SUPERCLASSES, and T is the root of the generalization hierarchy THING. Add node c where p is a parent class of I (p is already contained in the IDG of I). Add arc s from S to the EDG if s is contained within the path from at least one parent of I to some node in C .
3. Recursively expand each instance node within any attribute value set in the IDG into an EDG.

An example of an EDG is shown in Fig. 4.3.

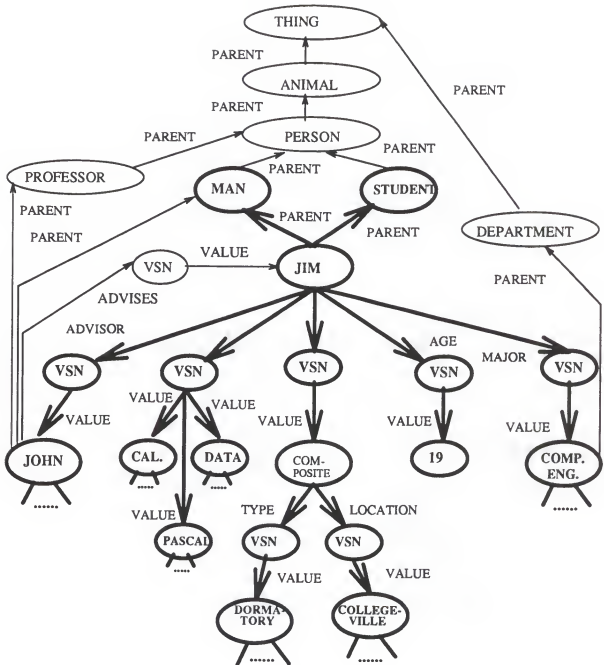


Figure 4.3. A graphical representation of the instance Jim

4.3.3 The Graph-Matching Technique

The intersection of two instances, $\text{INTERSECT}(I_1, I_2)$, is obtained by creating a new graph $EDG_c = \{V_c, E_c, R_c\}$. This graph is an intersection of the two extended description graphs of the two instances, I_1 and I_2 . The EDG_c is then converted into a class description as shown in the next subsection.

Consider

$$S = \{V_1 \times V_2, E_1 \times E_2, (R_1, R_2)\}$$

Then $\text{INTERSECT}(I_1, I_2) = EDG_c$ if there is a function F which maps EDG_c into S such that a component of EDG_c is mapped to a pair of matched components of EDG_1 and EDG_2 . The components of the EDG_c are generated as follows:

1. **Root.** The root of the EDG_c maps this matched pair of EDG_1 and EDG_2 .

That is,

$$F : R_c \rightarrow (R_1, R_2).$$

2. **Ancestors.** There exists a mapping between the ancestor nodes and arcs of EDG_c and those of EDG_1 and EDG_2 . In other words,

$$F : V_{pc} \rightarrow V_{p1} \times V_{p2}$$

$$F : V_{sc} \rightarrow V_{s1} \times V_{s2}$$

$$F : V_{vc} \rightarrow V_{v1} \times V_{v2}$$

There is an ancestor node and arc in EDG_c for each ancestor node and arc in both EDG_1 and EDG_2 . transitive closure of E_{p1} and E_{p2} guarantees that there is an arc from the root node of EDG_c to each of the most specific ancestor nodes in EDG_c . Moreover,

$$\text{IF } c \in V_{pc} \ \&$$

$$c_1 \in V_{p1} \ \&$$

$$c_2 \in V_{p2} \ \&$$

$$\text{CLASS NAME}(c) = \text{CLASS NAME}(c_1) = \text{CLASS NAME}(c_2) \ \&$$

\exists a path from R_1 to c_1 in EDG_1 &

\exists a path from R_2 to c_2 in EDG_2

THEN $F(c) = (c_1, c_2)$

Here $\text{CLASS NAME}(c)$ is the name of the class associated with the ancestor node c . For arcs:

IF $a = (c', c'') \in E_{pc}$ &

$a_1 = (c'_1, c''_1) \in E_{p_1}$ &

$a_2 = (c'_2, c''_2) \in E_{p_2}$ &

$F(c') = F(c'_1, c'_2)$ &

$F(c'') = F(c''_1, c''_2)$

THEN $F(a) = (a_1, a_2)$

To illustrate these concepts consider instance I_1 that has ancestors $V_{p_1} = \{\text{Student, Women, Person, Animal, Thing}\}$, and instance I_2 that has ancestors $V_{p_2} = \{\text{Child, Person, Animal, Thing}\}$. Then the intersection of these two instances is $V_{pc} = \{\text{Person, Animal, Thing}\}$.

3. **Attributes.** There exists a mapping between arcs in E_{ac} and a pair of arcs from E_{a_1} and E_{a_2} . That is,

$$F : E_{pc} \rightarrow E_{p_1} \times E_{p_2}$$

$$F : E_{ac} \rightarrow E_{a_1} \times E_{a_2}$$

$$F : E_{vc} \rightarrow E_{v_1} \times E_{v_2}$$

Every arc in E_{ac} is labeled using the most specific common attribute. Two attributes are related if they have a common ancestor in the attribute hierarchy such that

IF $a \in E_{ac}$ &

$a_1 \in E_{a_1}$ &

$a_2 \in E_{a_2}$ &
 \exists an attribute t :
 $\{L(a) = t$
 $\text{ANCESTOR}(t, L(a_1)) \&$
 $\text{ANCESTOR}(t, L(a_2)) \&$
 $\{\forall (\text{ANCESTOR}(y, L(a_1)) \& \text{ANCESTOR}(y, L(a_2))) \rightarrow$
 $\text{ANCESTOR}(y, t)\} \&$
 $t \neq \text{THING} \}$
 THEN $F(a) = (a_1, a_2)$.

Here, $\text{ANCESTOR}(t_1, t_2)$ is TRUE if the attribute t_1 is above attribute t_2 in the hierarchy. $L(a)$ is the attribute associated with attribute arc a .

To illustrate, consider the attributes of instance I_1 are $E_{a_1} = \{\text{Major, Course, Age, Advisor, Residence}\}$ and the attributes of instance I_2 are $\{\text{Title, Department, Teaches, Advises, Salary}\}$, then $E_{ac} = \{\text{Class}\}$ where Class is above Course and Teaches in the attribute hierarchy. $F(\text{Class}) = (\text{Course, Teaches})$.

4. **Attribute value sets.** An arc a in E_{ac} maps to a pair (a_1, a_2) from E_{a_1} and E_{a_2} . The arc a is associated with a value set resulting from the intersection of the value sets associated with a_1 and a_2 . First, the arc a is associated with a value set node, such that

IF $v \in V_{sc}$ &
 $v_1 \in V_{s_1}$ &
 $v_2 \in V_{s_2}$ &
 $a = (R_c, v)$ &
 $a_1 = (R_1, v_1)$ &
 $a_2 = (R_2, v_2)$ &
 $F(a) = (a_1, a_2)$ &

THEN $F(v) = (v_1, v_2)$

It is necessary to consider the intersection of every possible pair of values from the value set of a_1 and a_2 . This is called the *value set intersection* (VSI). VSI is defined as

$$\begin{aligned} \text{VSI}(a_1, a_2) = \{ (v, v_1, v_2) \parallel \exists v_1 \in \text{VS}(a_1) \ \& \ \exists v_2 \in \text{VS}(a_2) \ \& \\ \text{VINTERSECT}(v_1, v_2) = v \ \& \\ v \neq \text{NULL} \ \& \ (v, v_1, v_2) \text{ is distinct} \} \end{aligned}$$

VINTERSECT is the intersection of two value nodes and is defined in the next section. Consider two attributes a_1 and a_2 that are both called Course, and the following relationships hold:

$$\begin{aligned} \text{VS}(a_1) &= \{\text{Data Structures, Pascal, Calculus}\} \\ \text{VS}(a_2) &= \{\text{Probability, Physics}\} \\ \text{VINTERSECT}(\text{Data Structures, Probability}) &= \text{Course} \\ \text{VINTERSECT}(\text{Data Structures, Physics}) &= \text{Course} \\ \text{VINTERSECT}(\text{Pascal, Probability}) &= \text{Course} \\ \text{VINTERSECT}(\text{Pascal, Physics}) &= \text{Course} \\ \text{VINTERSECT}(\text{Calculus, Probability}) &= \text{Math Course} \\ \text{VINTERSECT}(\text{Calculus, Physics}) &= \text{Course} \\ \text{SUBSUME}(\text{Course, Math Course}) &= \text{TRUE} \end{aligned}$$

Then $\text{VSI}(a_1, a_2) = \{ \text{Course, Math Course} \} \ \& \ F(\text{Math Course}) = \{ \text{Calculus, Probability} \}.$

5. **Intersection of value nodes.** The intersection of two value nodes, $\text{VINTERSECT}(v_1, v_2)$ depends on their types as enumerated below:

- (a) Two atomic values (INTEGER, REAL, or STRING) have a non-NULL intersection only when they are the same type. If they are also the same value, then the intersection is the value. Otherwise, if they are the same type then the intersection is that type. The intersection of two atomic values of different types is NULL.
- (b) Two values of RANGE have a non-NULL intersection only if their ranges overlap. Thus the intersection of $\text{RANGE}(l_1, h_1)$ with $\text{RANGE}(l_2, h_2)$ is $\text{RANGE}(l_2, h_2)$ if $l_1 \leq l_2 \leq h_1$ and $l_2 \leq h_1 \leq h_2$. The intersection of $\text{RANGE}(l, h)$ with the integer n is n if $l \leq n \leq h$.
- (c) The intersection of two values of type INSTANCE or COMPOSITE is the INTERSECT of the values.

4.3.4 Class Description Generation From EDG

An EDG returned from the INTERSECT function is converted into a class description C as follows:

1. All classes created by a non-NULL intersection of instances are defined.
2. Remove all but most specific ancestor classes. Any ancestor class remaining which is on an arc from the root of EDG becomes an element of the SUPER-CLASSES of C.
3. Form an attribute restriction in C for each attribute arc from the root node of the EDG. SOME, EXACT, and ALL restrictions are created based on elements of the value set of the attribute as follows:
 - (a) If the value set of the attribute in EDG has n_1 values in $\langle v_1 \rangle$, n_2 values in $\langle v_2 \rangle$, and n_k values in $\langle v_k \rangle$ where v_1 is in the domain of v_2 , v_2 is in the domain of v_3, \dots, v_{k-1} is in the domain of v_k , then create a SOME $n_1 + n_2 + \dots + n_k \langle v_k \rangle$.

- (b) If a SOME $n \langle v \rangle$ restriction has been created, and a_1 and a_2 also have exactly n values each, change the SOME $n \langle v \rangle$ to EXACT $n \langle v \rangle$.
 - (c) If a SOME $n \langle v \rangle$ restriction has been created and all the values of a_1 and a_2 are in the domain of $\langle v \rangle$, but a_1 and a_2 do not have the same number of values, add an ALL $\langle v \rangle$.
4. C must then be classified correctly to determine all of its SUPERCLASSES, SUBCLASSES and INSTANCES.

Therefore, the intersection of the two instances Jim and Fred results in the following class description.

New_Class_ 1

SUPERCLASS: Student, Man

ATTRIBUTE RESTRICTIONS

Major: EXACT 1 CLASS Engineering

Courses: SOME 1 CLASS Math Course

SOME 2 CLASS Course

ALL CLASS Course

Age: EXACT 1 INSTANCE 19

Advisor: COMPOSITE

SUPERCLASS: Professor, Man

ATTRIBUTE RESTRICTIONS

Title: EXACT 1 Professor

Department: EXACT 1 CLASS Engineering

Teaches: SOME 1 CLASS Course

Advises: SOME 1 CLASS Student

Salary: EXACT 1 INTEGER

Residence: COMPOSITE

ATTRIBUTE RESTRICTIONS

Type: EXACT 1 CLASS Rental Unit

Location: EXACT 1 INSTANCE Collegeville

4.3.5 Schema Evolution

In this subsection we show how a new instance may be placed in a class if it fails to satisfy the necessary and sufficient conditions for class membership. As mentioned earlier, an instance I is either an exception and therefore an instance of a class or it is not an exception to a class but represents the beginning of a new class. If the first assumption is adopted, then it is necessary to alter the class structure to accommodate the new exception. A minimal requirement is described by the function EVOLVE. A new non-trivial class C' which results from the INTERSECT of I with other instances in the database such that:

1. $\text{EXTENSION}(C) \cap \text{EXTENSION}(C') \neq \text{NULL}$
2. $\text{EXTENSION}(C') - \text{EXTENSION}(C) = \{ I \}$

where $\text{EXTENSION}(C)$ is the set of instances which are members of class C . EVOLVE creates another new class C_n . $C_n = \text{INTERSECT}(I, \text{EXTENSION}(C))$. Note that C_n will necessarily subsume C and C' . The new class C_n represents the entity type which has been represented by C . As a new class evolves, it is subdivided into regions of similar instances. This results in a complex cluster of subclasses and instances characterized by localized areas of similarity.

4.4 Performance

A database management system based on CANDIDE and the conceptual clustering algorithm discussed in this chapter have been implemented in C on a UNIX workstation. Moreover, the C code has been ported to an 80836-based microcomputer running under protected mode by using an MS-DOS extender. The Computational complexity of the algorithm has not been studied formally, although there has been extensive work on SUBSUME. SUBSUME is intractable in the worst cases for all but

the most simplified data models. INTERSECT appears to be much more complex than SUBSUME in the worst case. Our implementation explores typical behavior for real applications to determine actual performance times.

Some benchmarks for CIMPLIES and INTERSECT were made using our current implementation on a 33mhz 80386-based microcomputer. We used a database of 400 ornamental plants in which each instance represents a particular plant species. Each instance had 25 attributes including height, growth rate, flower color, soil conditions and so forth. Using this database, COMPLIES can be evaluated at a rate of 100 instances per second. INTERSECT can be evaluated at a rate of 100 instances in 20 seconds. We expect this performance to improve with additional code optimization. In addition, it was not our intent at this point to produce an efficient algorithm, but rather to make the main point, namely, that a computer algorithm for generating database schemas ought to be based on modern theories of categorization. We intend to address the computational complexity issue with more efficient algorithm, in particular:

- Limit the number of intersections performed by introducing the notion of a *candidate set*. That is, INTERSECT will only be performed on those instances of the candidate set and not on all instances of the database. Moreover, an appropriate indexing scheme will be required for identifying and retrieving those instances of the candidate set.
- Develop an incremental version of INTERSECT in which not all intersections are computed at once, but only the most relevant ones first. The algorithm can be aborted at any time, or may be continued as long as needed until a satisfactory answer is obtained.

CHAPTER 5

IMPRECISE QUERYING

Beck et al [8] introduced a new semantic data model, CANDIDE, based on term-subsumption languages such as KL-ONE [12], in which classification and subsumption algorithms were utilized to process database queries. In CANDIDE, a query object is specified in the same notation as other database class objects. Thus, the data definition language (DDL) and the data manipulation language (DML) are identical and provide uniform treatment for all database objects. Query processing occurs by classifying the query object to determine its correct place in the class taxonomy. Classification is performed automatically through subsumption. A class C_1 subsumes C_2 if C_2 satisfies the necessary and sufficient conditions specified in the class description of C_1 . Essentially, the classifier finds the most specific classes that subsume the new class and the most general classes subsumed by the new class. A realization function is used to determine whether an instance meets an existing class description. Only those instances of the immediate superclasses are considered. Consequently, query processing is based on deductive inferencing about structured objects rather than procedural specification of operations. Inferencing techniques are defined formally by the subsumption function. Query specification is entirely declarative; the user need not specify *how* (in terms of algebraic operations) or from *where* (logical access paths and exact attribute names) the query is to be executed.

This type of query processing is *precise* in the sense that only those instances that totally satisfy the conditions stated in the query are retrieved. In the absence of such answers, that is, none of the instances of the database completely satisfy the conditions stated in the query, it is often desirable to retrieve instances that satisfy some if not all the conditions rather than return a null answer [3, 30, 39, 40, 41, 52]. Such is the case when requests are either intentionally or by necessity formulated in imprecise terms. For example, it may be advantageous to find the earliest airline

flights that most *closely* match a request submitted by a traveler, even though no available flights would completely match his request. With a database system that can only handle standard (specific) queries, the user must emulate such requests with standard queries [41]. Usually this means that the user is forced to retry a particular query with alternative values until satisfactory data are retrieved. If the user is unaware of any close alternatives, even this approach becomes infeasible. This aspect is of particular importance to naive users who have limited knowledge of the database. Moreover, queries reflect the presuppositions of users about the system and the information it contains [39]. With most query processors, queries that are based on erroneous presuppositions often result in null answers. These null answers are misleading since they do not point out these erroneous presuppositions. Null answers are disturbing because they do not guide or help the user in subsequent interactions as would be the case in human dialog, for example. More informative answers are required. User interfaces to database systems usually have limited rejection capabilities. Often the only rejections they are capable of, other than syntactic, are based on the schema. With an imprecise querying facility, the user-system interaction can be improved.

What it means for answers to be retrieved which are somehow 'close' to the specified query needs careful consideration. Expressions such as 'A is similar to B', or 'A is more similar to B than C', are common in our language, yet they are difficult to formalize. Traditionally, numerical methods have been used in which instances are expressed as points in n -dimensional space where similarity is expressed a distance between points in the space. Below a different approach is presented, in which similar instances are grouped together within a database taxonomy by using conceptual clustering. In the process, some problems with numerical methods are overcome while resulting in a notion of similarity which is more intuitive and natural.

Several researchers have addressed the issue of similarity between concepts [20, 25, 28, 47, 51]. Holyoak and Thagard [28] describe three different kinds of comparisons on the relationships between two concepts. Structural similarity refers to the components of objects. Two objects are structurally similar if mappings between components of the objects can be specified. Semantic similarity is a relationship between two concepts having the same meaning in the sense of sharing a common abstraction. Pragmatic similarity refers to the *context* in which a similarity comparison is applied. Context affects both the level of abstraction at which an object is considered and the relevance of its features. A query is one kind of specification of a context. The clustering algorithm is based on these types of comparisons.

The purpose of the clustering algorithm is to automatically position classes and instances in the generalization taxonomy. The algorithm consists of a deductive and inductive component. The deductive component assigns instances to existing classes and determines the superclasses of a new class based on satisfying necessary and sufficient conditions for class membership. The inductive component generates new class descriptions from the original query. This is done by generalizing the constraints specified in the query in different *directions* and to different *levels*. The clustering algorithm thus automatically generates a class taxonomy and groups instances into classes.

The result of the conceptual clustering process is not only a set of instances but also a schema that describes those data instances. This is an *intensional answer* to a query [31]. The resultant schema can be regarded as an 'abstracted answer' to the query. Conceptually, this abstracted answer is much more informative, in the sense that it is easier to comprehend by the user, than a set of instances. The distinct feature of this approach is that the intensional answer is specified in the data definition language (in the form of a schema) as opposed to predicates in other techniques. Therefore, we are able to utilize the full expressive power of the semantic data model to describe these instances.

This chapter is organized as follows. Section 2 presents a comparison of our approach to related work. Section 3 gives a detailed description of the algorithm in conjunction with an example. Performance issues are considered section 4.

5.1 Comparison to Related Work

Virtually all research work in the area of imprecision in database systems has been in the context of the relational data model [41]. The relational data model permits two kinds of imprecision: at the data value level and the tuple level. The former allows data values to be imprecise while the latter case there may be imprecision regarding the membership of a tuple in a particular relation. This imprecision is typically modeled in one of two ways: either using a probabilistic approach or a fuzzy one.

In the probabilistic approach, each candidate value is accompanied by a probability factor, a number between 0 and 1, specifying the probability that it is indeed the true value. The sum of these probabilities over all the candidate values is 1. Probabilistic data subsumes other types of data including precise data (probability of 1) and disjunctive data. Disjunctive data is a set of data elements to which the true value belongs. Each data element in this set has the same probability of being the true value.

The other major type of imprecise data is fuzzy data. The basic concept underlying fuzzy data is that of a fuzzy set. A fuzzy set A is a set of elements with an associated set membership function $u_A()$. The membership function assigns values, in the range 0 to 1, to each element of the set. This value indicates the grade of membership in the set. The greater the value the greater the degree of membership in the set. For example, we can define a fuzzy set *YOUNG* (denoting youth) as a set of ages 20, 30 and 40 with grades 0.9, 0.7, and 0.3, respectively.

To manipulate fuzzy databases, the standard relational algebra must be extended to fuzzy relations. Specifically, the imprecision in the values of the tuples creates

problems of value identification (e.g. in the join or in the removal of duplicates in the projections). Also, standard mathematical comparators such as $=$ or $<$ are superseded by fuzzy set comparators such as *similar-to* or *much-greater-than*. It is this type of fuzzy operators that allow the expression of fuzzy retrieval requests.

The FRDB system [52] is an experimental database system based on the ideas mentioned above. Each query in FRDB is a sequence of statements that perform fuzzy retrieval operations on the database. An optional argument of each statement is a threshold value, between 0 and 1, which is used to filter the tuples of the result by selecting only those whose grade values exceed that of the threshold. For example, in FRDB a user might specify a query 'select person, whose age is YOUNG, with threshold value of 0.8'. Here, *YOUNG* is a fuzzy set of values, person is a relation with attribute age, the domain of which is *YOUNG*. Thus, the user wants all persons whose ages have grades greater than 0.8.

The problem with this approach is that it associates a scale with abstract notions such as youth. What is this scale? How is youth measured? Moreover, will every user when formulating a query use the same scale? This becomes more acute in queries that reference more than one fuzzy set, such as 'select person, who is young and successful'. The underlying assumption in fuzzy retrieval systems is that all membership functions are commensurate: that is, they all use the same scale.

Alternative approaches to imprecise querying, as exemplified by ARES [30] and VAGUE [40], is to formulate a query solely on numerical measures of similarity. That is, distances are defined among elements of the domain. By dividing each distance by the diameter of the domain (the largest distance between two elements) a measure of *dissimilarity* is obtained. Its complement to 1 is a measure of similarity. Thus, two elements are similar if they are within a certain threshold. But how can these thresholds and distances (another type of scale) be attributed to abstract concepts? At the very least it is very subjective, and if the user is unaware of these thresholds and distances this approach becomes infeasible. In addition, this type of similarity

measure, as that of fuzzy sets, is context-free [36]. For example, by using this type of numerical similarity we would be able to classify objects as ‘tall’ if they exceeded a certain threshold. But, it is doubtful this threshold will be the same in the context of tall skyscrapers and tall people. Consequently, the fuzzy set *TALL* has to be explicitly defined for each type of physical object.

SEAVE [39] is a technique that extracts presuppositions from queries and verifies their correctness. It is useful here to distinguish between the capabilities to accommodate imprecise data and the ability to handle imprecise queries. Each of these capabilities can be provided separately. SEAVE is a technique that provides an imprecise querying facility for standard (precise) relational database systems. A component of SEAVE is a *supposition generalizer* that generalizes queries by weakening mathematical conditions or removing nonmathematical conditions altogether. It is assumed that three values are associated with each numeric attribute. These values are its upper bound, lower bound and a step value to specify the minimal weakening of a mathematical condition. For example, the attribute *SALARY* may have minimum 10000, maximum 50000 and step 1000. Therefore, a constraint such as *SALARY* < 40000 is generalized to *SALARY* < 41000. If an attribute is non-numeric, it is generalized by dropping the associated constraint. Thus, imprecise querying is achieved by continually generalizing the constraints in the query until a match is found. Notice how this approach is totally based on numerical methods. No provisions are given for nonnumeric attributes. In addition, the supposition generalizer weakens numeric constraints by arbitrary step values.

The basic idea underlying our conceptual clustering approach is that it generates classes that are complex conceptual clusters of instances rather than sets of instances that are within a certain numerical threshold. Conceptual clustering creates *conceptually cohesive* classes [36, 50]. Thus, instances of these classes exhibit structural (correspondence between object components), semantic (correspondences in meaning) and pragmatic (relevancy to the goal, query object) similarity.

CLUSTER/S [50] and COBWEB [20] are examples of systems that utilize this approach. CLUSTER/S is a conceptual clustering program that utilizes a goal-dependency network (GDN) to derive attributes related to the initial goal. For example, if the goal is 'Survive', then the concept 'Eat Food' is determined to be relevant through the GDN. Classes are formed that correspond to relevant attributes identified by the GDN. COBWEB uses a statistical measure (class utility measure) to evaluate the quality of different classes (clustering patterns) over the same set of instances. This class utility measure is designed to maximize the ability to predict instance attributes from knowledge of class membership. By using such a measure COBWEB would have difficulty creating classes that are relevant to a specified goal because pragmatic similarity is not considered. ACME [28] is a program that utilizes a connectionist approach to conceptual clustering. In this program constraints of a class are represented by means of a network of supporting and competing hypotheses regarding which elements of the instance to map. A cooperative algorithm for parallel constraint satisfaction identifies mapping hypotheses that collectively represent the overall mapping that best fits the interacting constraints. The network incorporated by ACME utilizes numerical semantic weights, thus a class modeled by this network is determined by a numerical measure of similarity.

5.2 Imprecise Querying Algorithm

The imprecise query algorithm is triggered by a null answer in response to a query. That is, if no instances in the database satisfy the conditions stated in the query the system will try to find instances that partially satisfy the query. The algorithm, as schematically described in Fig. 5.1, works as follows:

1. The original query is generalized by relaxing the attribute restrictions stated in the query. This is done according to the rules stated in subsection 3.2. Several new query objects can be generated.

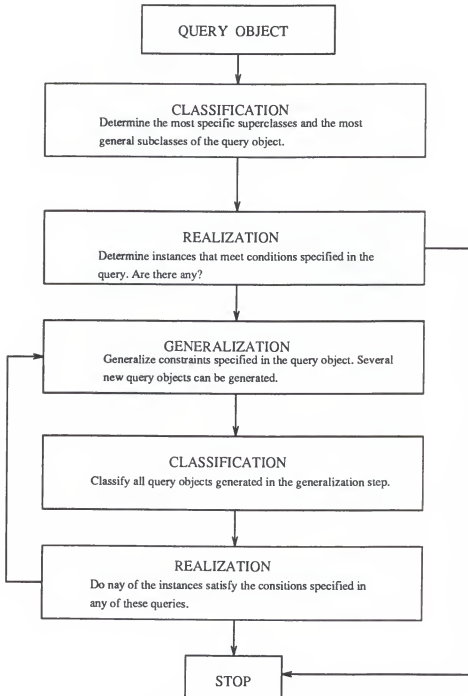


Figure 5.1. A schematic illustration of the imprecise querying algorithm

2. These generalized query objects are then classified in the taxonomy and the realization function is used to determine whether instances satisfy these query objects.
3. If no instances satisfy these generalized queries, the process is repeated for each generalized query until either all conditions are dropped (last step of generalization) or instances satisfy any of the queries.

Essentially, this technique follows up the failed query with several more general queries. This process is continued until a match is found.

Query generalization is accomplished by relaxing the attribute restrictions specified in the query. Each attribute restriction in CANDIDE possesses two types of constraints: a cardinality constraint and a value set constraint. For example, consider this simple query object with one attribute restriction

QUERY CLASS

SUPERCLASS: STUDENTS

ATTRIBUTE RESTRICTIONS

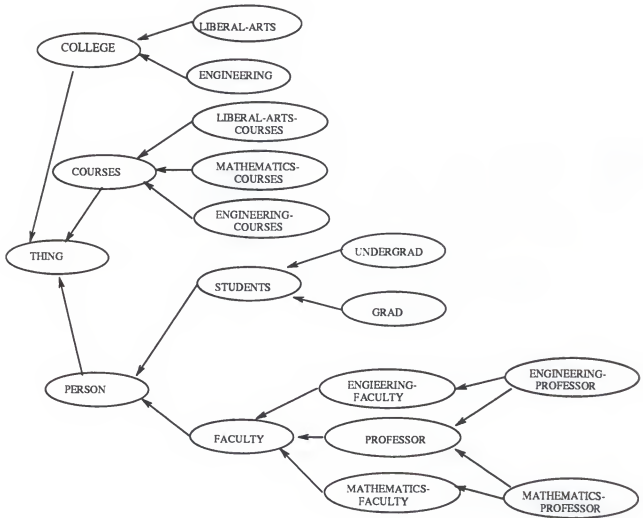
COURSE-LOAD: SOME 2 CLASS 'ENGINEERING-COURSE'

This query retrieves all students whose attribute COURSE-LOAD has at least two values (cardinality constraint) of the class 'ENGINEERING-COURSE' (value set constraint). Thus, an attribute restriction can be generalized in two possible directions: either by relaxing the cardinality constraint or the value set constraint. These two types of generalization detailed in the next two subsections.

5.2.1 Generalization of Cardinality Constraint

A cardinality constraint is generalized by applying the 'extension of quantification domain' rule [35]. Michaliski introduced the concept of a numerical quantifier, expressed in the form

$$\exists(I)v, S[v]$$



(a)



(b)

Figure 5.2. A CANDIDE class taxonomy and attribute hierarchy. a) Class taxonomy; b) Attribute hierarchy

where I , the index set, denotes a set of integers and $S[v]$ is an expression having v as a free variable. The previous expression evaluates to true if the number of values of v for which expression $S[v]$ is true is an element of the set I . Consequently, the above attribute restriction can be stated as

$$\exists(\geq 2)v, \{v \mid \text{attr-value}(\text{COURSE_LOAD}, v) \wedge v \in \text{'ENGINEERING_COURSE'}\}$$

Using the concept of the numerical quantifier, the extension of the quantification domain rule can be expressed as

$$\exists(I_1)v, S[v] \parallel < \exists(I_2)v, S[v], I_1 \subseteq I_2$$

where $\parallel <$ denotes a generalization. Notice how the domain of quantification was extended; the index set I_1 is a subset of I_2 . The three types of cardinality constraints modeled in CANDIDE are

SOME $n \langle v \rangle$

EXACT $n \langle v \rangle$

ALL $\langle v \rangle$

The SOME constraint means that there exists at least n attribute values, all of which are elements of the domain qualified by the value set constraint $\langle v \rangle$. The EXACT constraint denotes that the attribute must have exactly n values all of which satisfy the value set constraint $\langle v \rangle$. The ALL constraint specifies that all the value of the attribute must belong to the domain qualified by $\langle v \rangle$. Note the similarity of the ALL and SOME constraints to the universal and existential quantifiers of first order predicate calculus. These constraints are generalized as follows

1. EXACT $n \langle v \rangle \parallel <$ SOME $n \langle v \rangle$.

The cardinality constraint of exactly n values satisfying the value set constraint is generalized to *n or more* values satisfying the value set constraint. The index set of the EXACT constraint constrains only one element; $I_1 = \{x \mid x = n\}$.

The index of the generalized SOME constraint contains integers that are greater

than or equal to n ; $I_2 = \{x \mid x \geq n\}$. Furthermore, $I_2 \supseteq I_1$. This is equivalent to

$$\exists(n)v, S[v] \parallel < \exists(\geq n)v, S[v]$$

2. ALL $\langle v \rangle \parallel <$ SOME 1 $\langle v \rangle$.

This is equivalent to changing the universal quantifier to an existential quantifier. That is,

$$\forall v, S[v] \parallel < \exists v, S[v].$$

This generalization rule can be viewed as a special case of the extension of quantification domain rule.

3. Some $n \langle v \rangle \parallel <$ SOME $(n-1) \langle v \rangle, n > 1$.

In other words, the cardinality constraint of n or more values satisfying the value set constraint is generalized to $n - 1$ or more values satisfying that value set constraint. The index set of the first constraint is $I_1 = \{x \mid x \geq n\}$. The index set of the generalized constraint is $I_2 = \{x \mid x \geq (n - 1)\}$. Thus, $I_2 \supseteq I_1$. This is equivalent to

$$\exists(\geq n)v, S[v] \parallel < \exists(\geq (n - 1))v, S[v].$$

4. SOME 1 $\langle v \rangle$ can not be generalized any further by relaxing the cardinality constraint ($I_1 = \{x \mid x \geq 1\}$). It is either generalized by relaxing the value set constraint or the condition is dropped if the value set constraint can not be generalized any further.

Consequently, by relaxing the cardinality constraint of the above attribute restriction, the following attribute restriction is generated

COURSE-LOAD: SOME 1 CLASS 'ENGINEERING-COURSE'

5.2.2 Generalization of Value Set Constraints

A value set constraint is generalized by 'climbing the generalization hierarchy'. Essentially, the domain of the attribute restriction is generalized by substituting the value set constraint of the attribute with a domain that is more general, that of its

parent. These set-subset relationships are modeled in the attribute hierarchy and the class hierarchy of CANDIDE. An example of such hierarchies is shown in Fig. 5.2.

Value set constraints are generalized as follows:

1. $\text{dom}(\langle v \rangle) \parallel < \text{dom}(\text{parent}(\langle v \rangle))$.

If the domain of the value set constraint specified in an attribute restriction is a subset of the domain of the attribute in that restriction, the value set is generalized to that of its parent, as modeled in the class hierarchy.

2. $\text{dom}(\langle v \rangle) \parallel < \text{dom}(\text{parent}(\text{attr}))$.

If the domain of the value set restriction is equivalent to that of the attribute in that restriction, the attribute is superseded with its parent in attribute hierarchy.

3. If the parent of the attribute is 'THING', the condition is generalized by dropping the attribute restriction entirely.

Consider the following attribute restriction:

SALARY: EXACT 1 RANGE(30000, 40000)

This restriction states that the value of salary must be in the range of 30,000 to 40,000. Since, this range is a subset of the domain of SALARY, the value set constraint of the attribute restriction is generalized to that of the attribute SALARY. That is, the attribute restriction is generalized to

SALARY: EXACT 1 RANGE(20000, 60000).

As another example, consider the following attribute restriction:

CHAIRPERSON: SOME 1 CLASS 'PROFESSOR'.

This restriction states that there is at least one chairperson who is a professor. This restriction is generalized to

ADVISES: SOME 1 CLASS 'FACULTY'

because the domain of 'CHAIRPERSON' is 'PROFESSOR' and thus can not be

generalized any further. Hence, 'CHAIRPERSON' is replaced with its parent, 'ADVISES', the domain of which is class 'FACULTY'. Notice that 'FACULTY' subsumes and is a parent of 'PROFESSOR', as shown in Fig. 5.2a.

5.2.3 Example

To illustrate the algorithm consider the class and attribute hierarchies shown in Fig. 5.2 and the following query:

QUERY CLASS

SUPERCLASS: GRAD

ATTRIBUTE RESTRICTIONS

COURSE-LOAD: SOME 2 CLASS 'ENGINEERING-COURSE'

ADVISES: SOME 1 INSTANCE 'JACK SMITH'

that states 'retrieve all the graduate students who are registered for at least two engineering courses and one of their advisers is Jack Smith'. The definition of the instance 'Jack Smith' was given in section 3.1. Figures 5.3 illustrates the lattice of the generalized queries. Only the first level of this lattice is explicitly shown due to space requirements. All the generalized queries will necessarily subsume the original query. If none of the database instances satisfy these generalized queries, each of them will be generalized further until a match is found or all the conditions are dropped. At the first level, five new query objects were generated: Query-1, Query-2, Query-3, Query-4 and Query-5.

Query-1 and Query-2 were obtained by generalizing the first attribute restriction without modifying any other restriction. Query-1 was generated by relaxing the cardinality constraint of the first attribute restriction. This query would retrieve 'all graduate students who are registered in at least *one* engineering course and one of their advisers is Jack Smith'. Query-2 was obtained by generalizing the value set constraint of the first attribute restriction from the class of all engineering courses to the class of all courses. The class 'COURSES' subsumes 'ENGINEERING-COURSES',

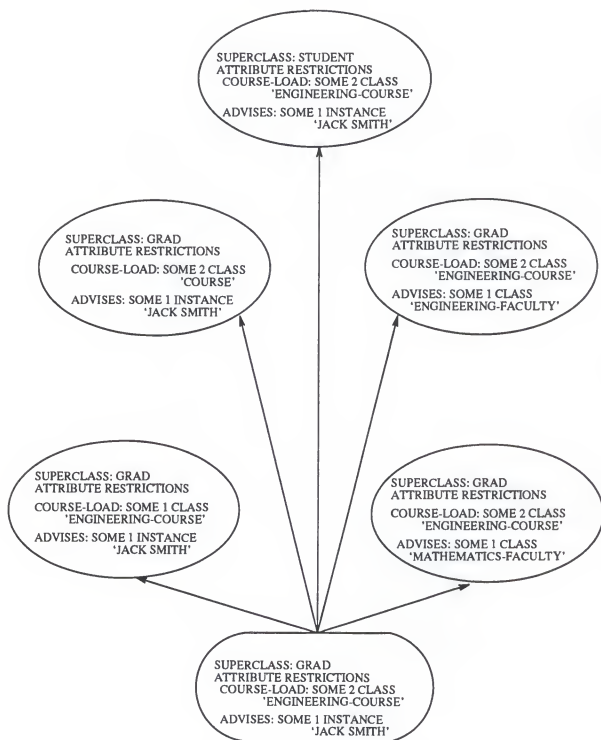


Figure 5.3. A lattice of generalized queries

as shown in Fig. 5.2a. Hence, Query-2 would retrieve ‘all graduate students who are registered for at least two *courses* and one of their advisers is Jack Smith’.

Query-3 was obtained by relaxing the superclass constraint. That is, instead of just considering graduate students, the query will be evaluated over all students. The class ‘STUDENTS’ subsumes ‘GRAD’ as shown in the class hierarchy of Fig. 5.2a. Query-3 would retrieve ‘all *students* who have registered for at least two engineering courses and one of their advisers is Jack Smith’.

The second attribute restriction can not be generalized by relaxing the cardinality constraint (SOME 1). However, by relaxing the value set constraint two queries were generated, Query-4 and Query-5. This is because two classes are the parents of Jack Smith: ‘MATHEMATICS-FACULTY’ and ‘ENGINEERING-FACULTY’. Thus, Query-4 will retrieve ‘all graduate students who are registered for at least two engineering courses and are advised by at least one *engineering faculty*’. Query-5, on the other hand, would retrieve ‘all graduate students who are registered for at least two engineering courses and are advised by at least one *mathematics faculty*’. Notice how the instance Jack Smith was generalized to the set of all mathematics faculty and the set of all engineering faculty, respectively.

5.3 Using the Database Taxonomy to Determine Relative Similarity

The problem of imprecise query processing can be stated as follows. Given a query, all instances in the database can be ordered by degree of similarity to the query. Any instance in the database can be construed to be similar in some way to the stated query (e.g in the trivial case, all instances are members of the universal class ‘THING’). The ordering is such that instances which match the query exactly are first, the most similar inexact matches are next, followed by inexact matches in decreasing order of similarity. The problem is to define a way of ordering similarity.

Given a database taxonomy, we can define a similarity measure based on the taxonomy. Simply put, instances are similar if they are members of the same class.

Conversely, database classes identify instances which are similar. Furthermore, the class provides an intensional description of precisely how the instances are similar. That is, the class contains attribute restrictions satisfied by all members of the class, and these attribute restrictions describe properties that all the member instances have in common.

The taxonomy can also be used to define a notion of relative similarity. Consider a set of instances with a common parent class. Starting with this parent class, moving up each level of the taxonomy locates classes which subsume the parent class. These classes are less restrictive (more general) than the parent class. Moreover, every instance of the parent class is an instance of any ancestor of the parent. Each ancestor has an extension which contains the extension of the original parent class as a subset. Each ancestor, going up the taxonomy, has an extension which expands the original set of instances. Each expansion includes more instances which are less similar to the original set. As a class becomes more inclusive (containing more instances) the less these instances will have in common. This is termed the *family resemblance effect* [47]. At the top of the taxonomy is the universal class, 'THING', which has as an extension the set of every instance in the database.

Consequently, how similar a generalized query is to the original query is determined by the number of generalizations performed. Consider a query object A and two other query objects B and C that are generalizations of A. Since these queries are generalizations of A, they will necessarily subsume A in the class taxonomy. Moreover, query B is more similar to query A than C if the number of generalization performed on A to generate B were less than those used to generate C. Thus, query B is more restrictive, by identifying more features by which its members are similar, than C. The more generalizations performed on a query the less restrictive and more inclusive it becomes. That is why the algorithm we have presented can be considered as a breadth- first search through the space of all possible query generalizations. The

algorithm evaluates queries in *decreasing* order of similarity to the original query; that is to say in *increasing* order of generalizations.

The advantage of using the taxonomy and the conceptual clustering algorithm to measure similarity is that no arbitrary numerical measures are needed. There are no thresholds. A ranking on similarity is provided by the depth in the taxonomy. Most important, the features of the objects are used to measure similarity. This includes the structural shapes of the objects, the semantic relationships between objects, and goals or pragmatic considerations which can influence the clustering schema.

5.4 Performance Issues

Imprecise querying involves additional computation. Clearly, if the query is attempted and matches some data, no additional computation is required. It is only when the query fails that more processing is required. The algorithm we have presented can be viewed as a *breadth-first search through the space of all possible query descriptions generated by the application of various generalization rules to the original query posed*. Hence, the performance of such an algorithm can be enhanced by constraining the number of queries generated and/or restricting the number of instances in the database over which the queries are evaluated. The number of queries generated by the algorithm can be constrained by imposing an upper limit on the level to which a query is generalized. By only considering a limited number of instances over which a query is evaluated, a *candidate set*, we not only restrict the number of instances over which query is evaluated but also certain types of generalizations. Consider the last example. If the candidate set was determined to be the class 'GRAD', Query-3 would not have been generated. This is because Query-3 is evaluated over 'STUDENT' which subsumes 'GRAD'.

Such limitations restrict the search space by not generating all possible, equally similar and relevant queries. A compromise must be reached between processing requirements and the quality of the answers generated.

CHAPTER 6 SUMMARY AND CONCLUSIONS

Semenatic data models are likely to play a major role in the coming years when dataabse management systems will be applied to store, retrieve and manage data from a vatiety of existing databases and non-database systems. The 'new' applications of databases in CAx systems (x: Design, Manufacturing, Publishing, Instruction, Software Engineering, etc) provide a challenge to sementic data modeling. A construct that transcends all semantic data models is that of a class. Class formation in semantic data models is ad hoc due to the varied treatment of classes and because the issue of grouping instances into classes is considered an art rather than a science. Currently, the process of generating class descriptions is very difficult due to the unpredictability and evolution of the real world and the error-prone process of interviewing end-users to determine relevant classes, attributes and relationships [10]. Moreover, the expressive power of semantic data models gives designers greater leeway in modeling an application domain without any guidance as to the appropriateness of the constructs used. This is termed *relativism* [26]. Therefore, the design process can be characterized by a certain amount of indeterminism inchoosing the constructs [11]. Consequently, the quality of the classes generated, and hence the schema are dependent on the designer's intuition and experience. In addition to this, it is extremely difficult, if not impossible, to define 'natural kinds' using necessary and sufficient conditions as utilized in some data models. Because of these reasons a database class is usually too general and does not specify all the pertinent information of the corresponding generic concept. Class definitions that do not accurately specify the generic concept modeled by that class lead to exceptions. The overwhelming majority of database systems are rigid in the sense that they do not tolerate exceptions. Exception handling duties are delegated to the user who is forced to *tailor the data to the schema* resulting in the possible loss of information.

In this thesis we have proposed a novel approach to class formation and schema design by the use of attribute-based purpose-directed conceptual clustering techniques. These techniques create *conceptually cohesive* [36] classes that are based on category theory rather than ad hoc classes. Schema generation occurs as a result of conceptually clustering the underlying data instances. Specifically, three conceptual clustering techniques have been introduced for schema integration, schema evolution and exception handling, and query processing. That is, all these three database functions have been considered as a form of schema design.

A conceptual clustering system for schema integration was detailed. This system generates views of different levels of abstraction and constraints based on the needs of different user groups. This technique employed by the system, a form of learning from observation, reasons at the instance level rather than at the class level, to generate views that reflect the actual data stored and to satisfy background knowledge. Classes generated by the system are conceptually cohesive and possibly overlapping, in contrast to other techniques that create only disjoint classes [50] and class descriptions of classes that pre-exist in the 'concept hierarchy' [13]. Unlike other systems, such as CLUSTER/2 [36], classes created by this technique are not ad hoc, but rather are constrained by their relevancy to the context (clustering seed). The system is capable of default reasoning by generating prototypical class descriptions. Moreover, the data model is of no consequence. Any semantic data model that supports classes, relationships between classes, an IS-A hierarchy and structural cardinality constraints can be used. The quality of the views generated are dependent on the background knowledge available to the system. Primarily, background knowledge models the interrelationships between attributes such as the synonym, set-subset, logical and mathematical interrelationships. It is these interrelationships that guide the clustering process and create more representative classes, unlike other techniques that do not capture these interrelationships and rely on literal similarity [37]. The technique presented here naturally lends itself to database integration and querying which is

discussed elsewhere [4]. Conceptual clustering tackles the problem of database integration through richer semantics. This is done by capturing the interrelationships between attributes and creating classes that are conceptually cohesive as compared to classes formed on the basis of thresholds [6, 27, 14]. Moreover, reasoning is done at the instance level rather than at the class level. But, database integration is a very difficult problem due to the structural and semantical diversities that must be dealt with. Further research is required to determine the suitability of this approach when dealing with such problems as incomplete and erroneous specifications and object correlation between databases.

An incremental semi-automatic conceptual clustering algorithm for schema evolution and exception handling was introduced. Specifically, the INTERSECT function generates a class description from common components of instances. The algorithm incorporates a number of features from category theory. In accordance with the family resemblance effect, the algorithm can generate a class with instances that have little or nothing in common. the algorithm can identify exceptions and modify the database schema to accommodate exceptions. The algorithm incorporates a tradeoff between cognitive models and similarity-based clustering. This is accomplished by combining explanation-based and case-based reasoning. The result is a data model with a more realistic treatment of classes and a better match between the resulting database schema and the application domain. Although the algorithm is semi-automatic rather than fully-automatic, it will act as a helpful assistant to a database designer. There is reason to suppose that a fully-automated algorithm is not possible. Our algorithm will begin to work on a database which has been constructed initially by hand, or by the algorithm introduced above. As the database develops and the schema becomes more complex, the algoirthm will be able to make better inferences.

Finally, an algorithm for processing imprecise queries by using conceptual clustering was presented. The algorithm contrasts with previous methods that use numerical

approaches in that query processing is achieved by deductive and inductive reasoning over object definitions rather than procedural specification of algebraic operations. Thus, the user may describe the query in terms which may be different from the exact terms in which the desired information is stored. In contrast to other techniques that rely on distance metrics and thresholds to process inexact queries, conceptual clustering retrieves instances that exhibit structural (correspondence between attributes), semantic (correspondence in meaning) and pragmatic (relevancy to the clustering seed) similarity. Query specification becomes entirely declarative in that the user does not specify how to execute the query. This responsibility is delegated to the system. This algorithm is a strict extension of the existing CANDIDE data model as compared to various other ad hoc implementations. Moreover, this technique provides an intensional answer to a query (a class taxonomy) that is conceptually more informative than a set of instances. This approach is being used on some applications, including an ornamental landscape plant selector. We are also applying this to clustering and retrieval of documents in an information retrieval system.

1875 (3)
12 11 1909

A. 12

1875 (3) 12 11 1909

REFERENCES

- [1] M.R. Anderberg. *Cluster Analysis for Applications*. Academic Press Inc., New York, 1973.
- [2] T.M. Anwar. An Implementation of the E-C-R Data Model with Semantic Integrity Constraints in Prolog. Master's Thesis, University of Florida, Gainesville, FL, 1987.
- [3] T.M. Anwar, H.W. Beck and S. B. Navathe. Knowledge Mining by Imprecise Querying: A Classification-Based Approach. In *Proceedings Eighth IEEE International Conference on Data Engineering*, Phoenix, AZ, February 1992.
- [4] T.M. Anwar, H.W. Beck and S.B. Navathe. Conceptual Clustering in Database Systems. In *Proceedings of the American Society for Information Systems Workshop on Classification Research*, Washington DC, Oct. 1991.
- [5] T.M. Anwar, S.B. Navathe and H.W. Beck. A Semantically Adaptive Modeling Interface for Schema Generation Over Multiple Databases. Technical Report TR-91-05, Database Systems Research and Development Center, University of Florida, Gainesville, FL, 1990.
- [6] C. Batini, M. Lenzirini and S.B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. In *ACM Computing Surveys*, Vol. 18, No. 4, pp. 323-363, Dec. 1986.
- [7] H.W. Beck, T.M. Anwar and S.B. Navathe. Towards Database Schema Generation by Conceptual Clustering. Technical Report TR-91-05, Database Systems Research and Development Center, University of Florida, Gainesville, FL, 1990.
- [8] H.W. Beck, S. Gala and S.B. Navathe. Classification as a Query Processing Technique in the CANDIDE Semantic Data Model. In *Proceedings of the IEEE Fifth International Conference on Data Engineering*, Los Angeles, CA, 1989.
- [9] F. Bergadano, S. Matwin, R.S. Michalski, and J. Zhang. Learning Two-tiered Descriptions of Flexible Concepts: The POSEIDON System, *Machine Learning Journal*, Vol. 7, pp. 73-96, May 1991.
- [10] A. Borgida, T. Mitchell and K.E. Williamson. Learning Improved Integrity Constraints and Schemas From Exceptions in Data and Knowledge Bases. In *Knowledge Base Management Systems*, Micheal L. Brodie and John Mylopoulos (eds.), Springer-Verlag, New York, 1986.
- [11] M. Bouzeghoub, G. Gardarin and E. Metais. Database Design Tools: An Expert System Approach. In *Proceedings of the 11th International Conference on Very Large Data Bases*, Stockholm, 1985.
- [12] R.J. Brachman and J.G. Schmolze. An Overview of KL-ONE Knowledge Representation System. *Cognitive Science*, Vol. 9, pp 171-216, 1985.

- [13] Y. Cai, N. Cercone and J. Han. An Attribute-Oriented Approach for Learning Classification Rules from Relational Databases. In *Proceedings of the IEEE Sixth International Conference on Data Engineering*, Los Angeles, Feb. 1990.
- [14] J. M. de Souza. SIS: A Schema Integration System. In *Proceedings of the Fifth British National Conference on Databases*, July, 1986.
- [15] F. Dretske. *Knowledge and the Flow of Information*. MIT Press, Cambridge, MA, 1981.
- [16] K. P. Eiselt. Toward a Unified Theory of Lexical Error Recovery. In *Proceedings of Cognitive Science Society*, Annual Meeting, 1991.
- [17] R. Elmasri, J. Larson, and S.B. Navathe. Schema Integration Algorithms for Federated Databases and Logical Database Design. Technical Report CSC-86-9:8212, Honeywell Computer Sciences Center, Bloomington, MN, 1986.
- [18] R. Elmasri, and S.B. Navathe. *Fundamentals of Database Systems*, Benjamin/Cummings Publishing Company, Menlo Park, CA, 1989.
- [19] R. Elmasri, J. Weeldreyer and A. Hevner. The Category Concept: An extension to the Entity-Relationship Model. In *International Journal on Data and Knowledge Engineering*, Vol. 1, p. 1, May 1985.
- [20] D. Fisher. Knowledge Acquisition Via Incremental Conceptual Clustering. In *Machine Learning*, Vol. 2, pp 139-172, 1987.
- [21] J. Fodor. *Psychosemantics*. Bradford Books, Cambridge MA, 1987.
- [22] R. Forsyth and R. Rada. *Machine Learning: Applications in Expert Systems And Information Systems*. Ellis Horwood Series in Artificial Intelligence, London, 1986.
- [23] S. French. *Decision Theory: An Introduction to the Mathematics of Rationality*. Ellis Horwood Series in Mathematics and its Applications, London, 1986.
- [24] P. Godfrey-Smith. Misinformation. *Canadian Journal of Philosophy*, Vol. 19, No. 4, pp. 533-550, 1988.
- [25] D. Gentner. Structure-Mapping: A Theoretical Framework for Analogy. *Cognitive Science*, Vol. 7, pp. 155-170, 1983.
- [26] M. Hammer and D. McLeod. Database Description with SDM: A Semantic Database Model. In *ACM Transactions on Database Systems*, Vol. 6, No. 3, Sept. 1981.
- [27] S. Hayne and S. Ram. Multi-User View Integration System: An Expert System for View Integration. In *Proceedings of the IEEE Sixth International Conference on Data Engineering*, Los Angeles, Feb. 1990.
- [28] K. Holyoak and P. Thagard. Analogical Mapping by Constraint Satisfaction. In *Cognitive Science*, Vol. 13, pp. 295-355, 1989.

- [29] R. Hull and R. King. Semantic Database Modeling: Survey, Applications and Research Issues. In *ACM Computing Surveys*, Vol. 19, p. 3. Sept. 1987.
- [30] T. Ichikawa and M. Hirakawa. ARES: A Relational Database with the Capability of Performing Flexible Interpretation of Queries. In *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 5, pp. 624-634, May 1986.
- [31] T. Imielinski. Intelligent Query Answering In Rule Based Systems. In *The Journal of Logic Programming*, Vol. 4, pp. 229-257, 1987.
- [32] J. Kolodner, editor. *Proceedings of a Workshop on Case-based Reasoning*, Morgan Kaufmann, San Mateo, CA, 1988.
- [33] P.F. Lazarsfeld and J.G. Reitz. *Toward a Theory of Applied Sociology*, AD 715659. Bur. Appl. Social Res., Columbia Univ., New York.
- [34] Q. Li and D. McLeod. Object Flavor Evolution through Learning in an Object-Oriented Database System. In *Proceedings of the ACM Conference on Office Information Systems*, Los Angeles, CA, 1988.
- [35] R.S. Michalski. A Theory and Methodology of Inductive Learning. In *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), Tioga, Palo Alto, CA, 1983.
- [36] R.S. Michalski and R.E. Stepp. Learning From Observation: Conceptual Clustering. In *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), Tioga, Palo Alto, CA, 1983.
- [37] G. Mineau and J. Gecsei. Structuring Knowledge Bases Using Automatic Learning. In *Proceedings of the IEEE Sixth International Conference on Data Engineering*, Los Angeles, Feb. 1990.
- [38] M.M.A. Morsi, S.B. Navathe and H.J. Kim. A Schema Management and Prototyping Interface for an Object-Oriented Database Environment. In *Proceedings of the IFIP working Group 8.1 Conference on The Object Oriented Approach in Information Systems*, Quebec City, Canada, North Holland, Amsterdam, 1991.
- [39] A. Motro. SEAVE: A Mechanism for Verifying User Presuppositions in Query Systems. In *ACM Transactions on Office Information Systems*, Vol. 4, No. 4, pp. 312-330, Oct. 1986.
- [40] A. Motro. VAGUE: a user interface to relational databases that permits vague queries. In *ACM Transactions on Office Information Systems*, Vol. 6, No. 3, pp. 187-214, July 1988.
- [41] A. Motro. Accommodating Imprecision in Database Systems. In *SIGMOD Record*, Vol. 19, No. 4, pp. 69-73, December 1990.
- [42] A. Motro and P. Buneman. Constructing Superviews. In the *Proceedings of the ACM Conference*, Ann Arbor, Michigan, 1981.
- [43] S.B. Navathe, R. Elmasri and J. Larson. Integrating User Views in Database Design. In *IEEE Computer*, Vol. 19, p. 1, Jan. 1986.

- [44] G.T. Nguyen and D. Rieu. Schema evolution in object-oriented systems. In *Data and Knowledge Engineering*, Vol. 4, Elsevier Science Publishers, New York, 1989.
- [45] J. R. Quinlan. Learning Efficient Classification Procedures and Their Application to Chess End Games. In *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), Tioga, Palo Alto, CA, 1983.
- [46] H.C. Romesberg. *Cluster Analysis for Researchers*. Lifetime Learning Publications, Belmont, CA, 1984.
- [47] E. Rosch. Human Categorization. In N. Warren (ed.), *Advances in Cross-Cultural Psychology*, Vol. 1, pp. 192-233, Academic Press, London, 1977.
- [48] J. M. Smith et al. MULTIBASE-Integrating Heterogeneous Distributed Database Systems. In the *Proceedings of the National Computer Conference*, 1981.
- [49] J.F. Sowa. *Conceptual Structures*. Addison-Wesley Publishing Company, Reading, MA, 1984.
- [50] R.E. Stepp and R.S. Michalski. Conceptual Clustering: Inventing Goal-Oriented Classifications of Structured Objects. In *Machine Learning: An Artificial Intelligence Approach*, Vol. 2, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), Tioga, Palo Alto, CA, 1986.
- [51] A. Tversky. Features of Similarity. In *Psychological Review*, Vol. 84, No. 4, pp. 327-352, July 1977.
- [52] M. Zemankova and A. Kandel. Implementing imprecision in information systems. In *Information Sciences*, Vol. 37, No. 1, 2, 3, pp. 107-141, Dec. 1985.

BIOGRAPHICAL SKETCH

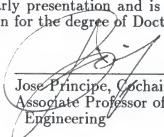
I was born in Egypt on July 25th, 1964 and attended school in Egypt, Kuwait and the United States. In 1985 I attained the degree of Bachelor of Science in Electrical Engineering from the University of Texas at Austin. Subsequently, I enrolled at the University of Florida and acquired the degree of Master of Science in the same field two years later. I have since worked in a software company in Kuwait where we developed specific software products for the Arabic language. I joined the Ph.D program at the University of Florida in the Fall of 1988.

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.




Shamkant B. Navathe, Chair
Professor of Electrical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



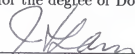
Jose Principe, Cochair
Associate Professor of Electrical
Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



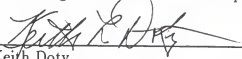
Sharma Chakravorthy
Associate Professor of Computer and
Information Sciences

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



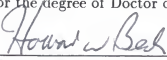
Herman Lam
Associate Professor of Electrical
Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



Keith Doty
Professor of Electrical Engineering

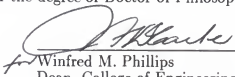
I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



Howard W. Beck
Assistant Professor of Agricultural
Engineering

This dissertation was submitted to the Graduate Faculty of the College of Engineering and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

August 1992



for Winfred M. Phillips
Dean, College of Engineering

Madelyn M. Lockhart
Dean, Graduate School